Clustering Billions of Reads for DNA Data Storage

Cyrus Rashtchian[a,b] **Konstantin Makarychev**[a,c] **Miklós Rácz**[a,d] **Siena Dumas Ang**[a]
Djordje Jevdjic[a] **Sergey Yekhanin**[a] **Luis Ceze**[a,b] **Karin Strauss**[a]
[a]Microsoft Research, [b]CSE at University of Washington,
[c]EECS at Northwestern University, [d]ORFE at Princeton University

Abstract

Storing data in synthetic DNA offers the possibility of improving information density and durability by several orders of magnitude compared to current storage technologies. However, DNA data storage requires a computationally intensive process to retrieve the data. In particular, a crucial step in the data retrieval pipeline involves clustering billions of strings with respect to edit distance. Datasets in this domain have many notable properties, such as containing a very large number of small clusters that are well-separated in the edit distance metric space. In this regime, existing algorithms are unsuitable because of either their long running time or low accuracy. To address this issue, we present a novel distributed algorithm for approximately computing the underlying clusters. Our algorithm converges efficiently on any dataset that satisfies certain separability properties, such as those coming from DNA data storage systems. We also prove that, under these assumptions, our algorithm is robust to outliers and high levels of noise. We provide empirical justification of the accuracy, scalability, and convergence of our algorithm on real and synthetic data. Compared to the state-of-the-art algorithm for clustering DNA sequences, our algorithm simultaneously achieves higher accuracy and a 1000x speedup on three real datasets.

1 Introduction

Existing storage technologies cannot keep up with the modern data explosion. Thus, researchers have turned to fundamentally different physical media for alternatives. Synthetic DNA has emerged as a promising option, with theoretical information density of multiple orders of magnitude more than magnetic tapes [12, 24, 26, 52]. However, significant biochemical and computational improvements are necessary to scale DNA storage systems to read/write exabytes of data within hours or even days.

Encoding a file in DNA requires several preprocessing steps, such as randomizing it using a pseudo-random sequence, partitioning it into hundred-character substrings, adding address and error correction information to these substrings, and finally encoding everything to the $\{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}\}$ alphabet. The resulting collection of short strings is synthesized into DNA and stored until needed.

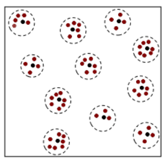
Figure 1: DNA storage datasets have many small clusters that are well-separated in edit distance.

To retrieve the data, the DNA is accessed using next-generation sequencing, which results in several noisy copies, called *reads*, of each originally synthesized short string, called a *reference*. With current technologies, these references and reads contain hundreds of characters, and in the near future, they will likely contain thousands [52]. After sequencing, the goal is to recover the unknown references from the observed reads. The first step, which is the focus of this paper, is to cluster the reads into groups, each of which is the set of noisy copies of a single reference.

The output of clustering is fed into a consensus-finding algorithm, which predicts the most likely reference to have produced each cluster of reads. As Figure 1 shows, datasets typically contain only a handful of reads for each reference, and each of these reads differs from the reference by insertions, deletions, and/or substitutions. The challenge of clustering is to achieve high precision and recall of many small underlying clusters, in the presence of such errors.

Datasets arising from DNA storage have two striking properties. First, the number of clusters grows linearly with the input size. Each cluster typically consists of five to fifteen noisy copies of the same reference. Second, the clusters are separated in edit distance, by design (via randomization). We investigate approximate clustering algorithms for large collections of reads with these properties.

Suitable algorithms must satisfy several criteria. First, they must be distributed, to handle the billions of reads coming from modern sequencing machines. Second, their running time must scale favorably with the number of clusters. In DNA storage datasets, the size of the clusters is fixed and determined by the number of reads needed to recover the data. Thus, the number of clusters k grows linearly with the input size n (i.e., $k = \Omega(n)$). Any methods requiring $\Omega(k \cdot n) = \Omega(n^2)$ time or communication would be too slow for billion-scale datasets. Finally, algorithms must be robust to noise and outliers, and they must find clusters with relatively large diameters (e.g., linear in the dimensionality).

These criteria rule out many clustering methods. Algorithms for k-medians and related objectives are unsuitable because they have running time or communication scaling with $k \cdot n$ [19, 29, 33, 42]. Graph clustering methods, such as correlation clustering [4, 9, 18, 47], require a similarity graph.[1] Constructing this graph is costly, and it is essentially equivalent to our clustering problem, since in DNA storage datasets, the similarity graph has connected components that are precisely the clusters of noisy reads. Linkage-based methods are inherently sequential, and iteratively merging the closest pair of clusters takes quadratic time. Agglomerative methods that are robust to outliers do not extend to versions that are distributed and efficient in terms of time, space, and communication [2, 8].

Turning to approximation algorithms, tools such as metric embeddings [43] and locality sensitive hashing (LSH) [31] trade a small loss in accuracy for a large reduction in running time. However, such tools are not well understood for edit distance [16, 17, 30, 38, 46], even though many methods have been proposed [15, 27, 39, 48, 54]. In particular, no published system has demonstrated the potential to handle billions of reads, and no efficient algorithms have experimental or theoretical results supporting that they would achieve high enough accuracy on DNA storage datasets. This is in stark contrast to set similarity and Hamming distance, which have many positive results [13, 36, 40, 49, 55].

Given the challenges associated with existing solutions, we ask two questions: (1) Can we design a distributed algorithm that converges in sub-quadratic time for DNA storage datasets? (2) Is it possible to adapt techniques from metric embeddings and LSH to cluster billions of strings in under an hour?

Our Contributions We present a distributed algorithm that clusters billions of reads arising from DNA storage systems. Our agglomerative algorithm utilizes a series of filters to avoid unnecessary distance computations. At a high level, our algorithm iteratively merges clusters based on random representatives. Using a hashing scheme for edit distance, we only compare a small subset of representatives. We also use a light-weight check based on a binary embedding to further filter pairs. If a pair of representatives passes these two tests, edit distance determines whether the clusters are merged. Theoretically and experimentally, our algorithm satisfies four desirable properties.

Scalability: Our algorithm scales well in time and space, in shared-memory and shared-nothing environments. For n input reads, each of P processors needs to hold only $O(n/P)$ reads in memory.

Accuracy: We measure accuracy as the fraction of clusters with a majority of found members and no false positives. Theoretically, we show that the separation of the underlying clusters implies our algorithm converges quickly to a correct clustering. Experimentally, a small number of communication rounds achieve 98% accuracy on multiple real datasets, which suffices to retrieve the stored data.

Robustness: For separated clusters, our algorithm is optimally robust to adversarial outliers.

Performance: Our algorithm outperforms the state-of-the-art clustering method for sequencing data, Starcode [57], achieving higher accuracy with a 1000x speedup. Our algorithm quickly recovers clusters with large diameter (e.g., 25), whereas known string similarity search methods perform poorly with distance threshold larger than four [35, 53]. Our algorithm is simple to implement in any distributed framework, and it clusters 5B reads with 99% accuracy in 46 minutes on 24 processors.

[1]The similarity graph connects all pairs of elements with distance below a given threshold.

1.1 Outline

The rest of the paper is organized as follows. We begin, in Section 2, by defining the problem statement, including clustering accuracy and our data model. Then, in Section 3, we describe our algorithm, hash function, and binary signatures. In Section 4, we provide an overview of the theoretical analysis, with most details in the appendix. In Section 5, we empirically evaluate our algorithm. We discuss related work in Section 6 and conclude in Section 7.

2 DNA Data Storage Model and Problem Statement

For an alphabet Σ, the *edit distance* between two strings $x, y \in \Sigma^*$ is denoted $d_E(x, y)$ and equals the minimum number of insertions, deletions, or substitutions needed to transform x to y. It is well known that d_E defines a metric. We fix $\Sigma = \{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}\}$, representing the four DNA nucleotides. We define the distance between two nonempty sets $C_1, C_2 \subseteq \Sigma^*$ as $d_E(C_1, C_2) = \min_{x \in C_1, y \in C_2} d_E(x, y)$. A *clustering* $\mathbf{C}$ of a finite set $S \subseteq \Sigma^*$ is any partition of S into nonempty subsets.

We work with the following definition of accuracy, motivated by DNA storage data retrieval.

Definition 2.1 (Accuracy)**.** Let $\mathbf{C}, \widetilde{\mathbf{C}}$ be clusterings. For $1/2 < \gamma \leqslant 1$ the *accuracy* of $\widetilde{\mathbf{C}}$ with respect to $\mathbf{C}$ is

$$\mathcal{A}_\gamma(\mathbf{C}, \widetilde{\mathbf{C}}) = \max_{\pi} \frac{1}{|\mathbf{C}|} \sum_{i=1}^{|\mathbf{C}|} \mathbf{1}\{\widetilde{C}_{\pi(i)} \subseteq C_i \text{ and } |\widetilde{C}_{\pi(i)} \cap C_i| \geqslant \gamma |C_i|\},$$

where the max is over all injective maps $\pi : \{1, 2, \ldots, |\widetilde{\mathbf{C}}|\} \to \{1, 2, \ldots, \max(|\mathbf{C}|, |\widetilde{\mathbf{C}}|)\}$.

We think of $\mathbf{C}$ as the underlying clustering and $\widetilde{\mathbf{C}}$ as the output of an algorithm. The accuracy $\mathcal{A}_\gamma(\mathbf{C}, \widetilde{\mathbf{C}})$ measures the number of clusters in $\widetilde{\mathbf{C}}$ that overlap with some cluster in $\mathbf{C}$ in at least a γ-fraction of elements while containing no false positives.[2] This is a stricter notion than the standard classification error [8, 44]. Notice that our accuracy definition does not require that the clusterings be of the same set. We will use this to compare clusterings of S and $S \cup \mathsf{O}$ for a set of outliers $\mathsf{O} \subseteq \Sigma^*$.

For DNA storage datasets, the underlying clusters have a natural interpretation. During data retrieval, several molecular copies of each original DNA strand (reference) are sent to a DNA sequencer. The output of sequencing is a small number of noisy reads of each reference. Thus, the reads that correspond to the same reference form a cluster. This interpretation justifies the need for high accuracy: each underlying cluster represents one stored unit of information.

Data Model To aid in the design and analysis of clustering algorithms for DNA data storage, we introduce the following natural generative model. First, pick many random centers (representing original references), then perturb each center by insertions, deletions, and substitutions to acquire the elements of the cluster (representing the noisy reads). We model the original references as random strings because during the encoding process, the original file has been randomized using a fixed pseudo-random sequence [45]. We make this model precise, starting with the perturbation.

Definition 2.2 (p-noisy copy)**.** For $p \in [0, 1]$ and $z \in \Sigma^*$, define a p-noisy copy of z by the following process. For each character in z, independently, do one of the following four operations: (i) keep the character unchanged with probability $(1 - p)$, (ii) delete it with probability $p/3$, (iii) with probability $p/3$, replace it with a character chosen uniformly at random from Σ, or (iv) with probability $p/3$, keep the character and insert an additional one after it, chosen uniformly at random from Σ.

We remark that our model and analysis can be generalized to incorporate separate deletion, insertion, and substitution probabilities $p = p_D + p_I + p_S$, but we use balanced probabilities $p/3$ to simplify the exposition. Now, we define a noisy cluster. For simplicity, we assume uniform cluster sizes.

Definition 2.3 (Noisy cluster of size s)**.** We define the distribution $\mathcal{D}_{s,p,m}$ with cluster size s, noise rate $p \in [0, 1]$, and dimension m. Sample a cluster $C \sim \mathcal{D}_{s,p,m}$ as follows: pick a center $z \in \Sigma^m$ uniformly at random; then, each of the s elements of C will be an independent p-noisy copy of z.

With our definition of accuracy and our data model in hand, we define the main clustering problem.

[2] The requirement $\gamma \in (1/2, 1]$ implies $\mathcal{A}_\gamma(\mathbf{C}, \widetilde{\mathbf{C}}) \in [0, 1]$.

Problem Statement Fix p, m, s, n. Let $\mathbf{C} = \{C_1, \ldots, C_k\}$ be a set of $k = n/s$ independent clusters $C_i \sim \mathcal{D}_{s,p,m}$. Given an accuracy parameter $\gamma \in (1/2, 1]$ and an error tolerance $\varepsilon \in [0, 1]$, on input set $S = \cup_{i=1}^{k} C_i$, the goal is to quickly find a clustering $\widetilde{\mathbf{C}}$ of S with $\mathcal{A}_\gamma(\mathbf{C}, \widetilde{\mathbf{C}}) \geqslant 1 - \varepsilon$.

3 Approximately Clustering DNA Storage Datasets

Our distributed clustering method iteratively merges clusters with similar representatives, alternating between local clustering and global reshuffling. At the core of our algorithm is a hash family that determines (i) which pairs of representatives to compare, and (ii) how to repartition the data among the processors. On top of this simple framework, we use a cheap pre-check, based on the Hamming distance between binary signatures, to avoid many edit distance comparisons. Our algorithm achieves high accuracy by leveraging the fact that DNA storage datasets contain clusters that are well-separated in edit distance. In this section, we will define separated clusterings, explain the hash function and the binary signature, and describe the overall algorithm.

3.1 Separated Clusters

The most important consequence of our data model $\mathcal{D}_{s,p,m}$ is that the clusters will be well-separated in the edit distance metric space. Moreover, this reflects the actual separation of clusters in real datasets. To make this precise, we introduce the following definition.

Definition 3.1. A clustering $\{C_1, \ldots, C_k\}$ is (r_1, r_2)-*separated* if C_i has diameter[3] at most r_1 for every $i \in \{1, 2, \ldots, k\}$, while any two different clusters C_i and C_j satisfy $d_E(C_i, C_j) > r_2$.

DNA storage datasets will be separated with $r_2 \gg r_1$. Thus, recovering the clusters corresponds to finding pairs of strings with distance at most r_1. Whenever $r_2 \geqslant 2 \cdot r_1$, our algorithm will be robust to outliers. In Section 4, we provide more details about separability under our DNA storage data model. We remark that our clustering separability definition differs slightly from known notions [2, 3, 8] in that we explicitly bound both the diameter of clusters and distance between clusters.

3.2 Hashing for Edit Distance

Algorithms for string similarity search revolve around the simple fact that when two strings $x, y \in \Sigma^m$ have edit distance at most r, then they share a substring of length at least $m/(r + 1)$. However, insertions and deletions imply that the matching substrings may appear in different locations. Exact algorithms build inverted indices to find matching substrings, and many optimizations have been proposed to exactly find all close pairs [34, 51, 57]. Since we need only an approximate solution, we design a hash family based on finding matching substrings quickly, without being exhaustive. Informally, for parameters w, ℓ, our hash picks a random "anchor" a of length w, and the hash value for x is the substring of length $w + \ell$ starting at the first occurrence of a in x.

We formally define the family of hash functions $\mathcal{H}_{w,\ell} = \{h_{\pi,\ell} : \Sigma^* \to \Sigma^{w+\ell}\}$ parametrized by w, ℓ, where π is a permutation of Σ^w. For $x = x_1 x_2 \cdots x_m$, the value of $h_{\pi,\ell}(x)$ is defined as follows. Find the earliest, with respect to π, occurring w-gram a in x, and let i be the index of the first occurrence of a in x. Then, $h_{\pi,\ell}(x) = x_i \cdots x_{m'}$ where $m' = \min(m, i + w + \ell)$. To sample $h_{\pi,\ell}$ from $\mathcal{H}_{w,\ell}$, simply pick a uniformly random permutation $\pi : \Sigma^w \to \Sigma^w$.

Note that $\mathcal{H}_{w,\ell}$ resembles MinHash [13, 14] with the natural mapping from strings to sets of substrings of length $w + \ell$. Our hash family has the benefit of finding long substrings (such as $w + \ell = 16$), while only having the overhead of finding anchors of length w. This reduces computation time, while still leading to effective hashes. We now describe the signatures.

3.3 Binary Signature Distance

The q-gram distance is an approximation for edit distance [50]. By now, it is a standard tool in bioinformatics and string similarity search [27, 28, 48, 54]. A q-gram is simply a substring of length q, and the q-gram distance measures the number of different q-grams between two strings. For a string

[3]A cluster C has diameter at most r if $d_E(x, y) \leqslant r$ for all pairs $x, y \in C$.

Algorithm 1 Clustering DNA Strands

```
1: function CLUSTER($S$, $r$, $q$, $w$, $\ell$, $\theta_{low}$, $\theta_{high}$, comm_steps, local_steps)
2:     $\widetilde{\mathbf{C}} = S$.
3:     For $i = 1, 2, \ldots,$ comm_steps:
4:         Sample $h_{\pi,\ell} \sim \mathcal{H}_{w,\ell}$ and hash-partition clusters, applying $h_{\pi,\ell}$ to representatives.
5:         For $j = 1, 2, \ldots,$ local_steps:
6:             Sample $h_{\pi,\ell} \sim \mathcal{H}_{w,\ell}$.
7:             For $C \in \widetilde{\mathbf{C}}$, sample a representative $x_C \sim C$, and then compute the hash $h_{\pi,\ell}(x_C)$.
8:             For each pair $x, y$ with $h_{\pi,\ell}(x) = h_{\pi,\ell}(y)$:
9:                 If $(d_H(\sigma(x), \sigma(y)) \leqslant \theta_{low})$ or $(d_H(\sigma(x), \sigma(y)) \leqslant \theta_{high}$ and $d_E(x, y) \leqslant r)$:
10:                    Update $\widetilde{\mathbf{C}} = (\widetilde{\mathbf{C}} \setminus \{C_x, C_y\}) \cup \{C_x \cup C_y\}$.
11:    return $\widetilde{\mathbf{C}}$.
12: end function
```

$x \in \Sigma^m$, let the binary signature $\sigma_q(x) \in \{0,1\}^{4^q}$ be the indicator vector for the set q-grams in x. Then, the q-gram distance between x and y equals the Hamming distance $d_H(\sigma_q(x), \sigma_q(y))$.

The utility of the q-gram distance is that the Hamming distance $d_H(\sigma_q(x), \sigma_q(y))$ approximates the edit distance $d_E(x, y)$, yet it is much faster to check $d_H(\sigma_q(x), \sigma_q(y)) \leqslant \theta$ than to verify $d_E(x, y) \leqslant r$. The only drawback of the q-gram distance is that it may not faithfully preserve the separation of clusters, in the worst case. This implies that the q-gram distance by itself is not sufficient for clustering. Therefore, we use binary signatures as a coarse filtering step, but reserve edit distance for ambiguous merging decisions. We provide theoretical bounds on the q-gram distance in Section 4.1 and Appendix B. We now explain our algorithm.

3.4 Algorithm Description

We describe our distributed, agglomerative clustering algorithm (displayed in Algorithm 1). The algorithm ingests the input set $S \subset \Sigma^*$ in parallel, so each core begins with roughly the same number of reads. Signatures $\sigma_q(x)$ are pre-computed and stored for each $x \in S$. The clustering $\widetilde{\mathbf{C}}$ is initialized as singletons. It will be convenient to use the notation x_C for an element $x \in C$, and the notation C_x for the cluster that x belongs to. We abuse notation and use $\widetilde{\mathbf{C}}$ to denote the current global clustering. The algorithm alternates between global communication and local computation.

Communication One representative x_C is sampled uniformly from each cluster C_x in the current clustering $\widetilde{\mathbf{C}}$, in parallel. Then, using shared randomness among all cores, a hash function $h_{\pi,\ell}$ is sampled from $\mathcal{H}_{w,\ell}$. Using this same hash function for each core, a hash value is computed for each representative x_C for cluster C in the current clustering $\widetilde{\mathbf{C}}$. The communication round ends by redistributing the clusters randomly using these hash values. In particular, the value $h_{\pi,\ell}(x_c)$ determines which core receives C. The current clustering $\widetilde{\mathbf{C}}$ is thus repartitioned among cores.

Local Computation The local computation proceeds independently on each core. One local round revolves around one hash function $h_{\pi,\ell} \sim \mathcal{H}_{w,\ell}$. Let $\widetilde{\mathbf{C}}_j$ be the set of clusters that have been distributed to the jth core. During each local clustering step, one uniform representative x_C is sampled for each cluster $C \in \widetilde{\mathbf{C}}_j$. The representatives are bucketed based on $h_{\pi,\ell}(x_c)$. Now, the local clustering requires three parameters, $r, \theta_{low}, \theta_{high}$, set ahead of time, and known to all the cores. For each pair y, z in a bucket, first the algorithm checks whether $d_H(\sigma_q(y), \sigma_q(z)) \leqslant \theta_{low}$. If so, the clusters C_y and C_z are merged. Otherwise, the algorithm checks if both $d_H(\sigma_q(y), \sigma_q(z)) \leqslant \theta_{high}$ and $d_E(x, y) \leqslant r$, and merges the clusters C_y and C_z if these two conditions hold. Immediately after a merge, $\widetilde{\mathbf{C}}_j$ is updated, and C_x corresponds to the present cluster containing x. Note that distributing the clusters among cores during communication implies that no coordination is needed after merges. The local clustering repeats for *local_steps* rounds before moving to the next communication round.

Termination After the local computation finishes, after the last of *comm_steps* communication rounds, the algorithm outputs the current clustering $\widetilde{\mathbf{C}} = \bigcup_j \widetilde{\mathbf{C}}_j$ and terminates.

4 Theoretical Algorithm Analysis

4.1 Cluster Separation and Binary Signatures

When storing data in DNA, the encoding process leads to clusters with nearly-random centers. Recall that we need the clusters to be far apart for our algorithm to perform well. Fortunately, random cluster centers will have edit distance $\Omega(m)$ with high-probability. Indeed, two independent random strings have expected edit distance $c_{\mathsf{ind}} \cdot m$, for a constant $c_{\mathsf{ind}} > 0$. Surprisingly, the exact value of c_{ind} remains unknown. Simulations suggest that $c_{\text{ind}} \approx 0.51$, and it is known that $c_{\text{ind}} > 0.338$ [25].

When recovering the data, DNA storage systems receive clusters that consist of p-noisy copies of the centers. In particular, two reads inside of a cluster will have edit distance $O(pm)$, since they are p-noisy copies of the same center. Therefore, any two reads in different clusters will be far apart in edit distance whenever $p \ll c_{\mathsf{ind}}$ is a small enough constant. We formalize these bounds and provide more details, such as high-probability results, in Appendix A.

Another feature of our algorithm is the use of binary signatures. To avoid incorrectly merging distinct clusters, we need the clusters to be separated according to q-gram distance. We show that random cluster centers will have q-gram distance $\Omega(m)$ when $q = 2\log_4 m$. Additionally, for any two reads x, y, we show that $d_H(\sigma_q(x), \sigma_q(y)) \leqslant 2q \cdot d_E(x, y)$, implying that if x and y are in the same cluster, then their q-gram distance will be at most $O(qpm)$. Therefore, whenever $p \ll 1/q \approx 1/\log m$, signatures will already separate clusters. For larger p, we use the pair of thresholds $\theta_{low} < \theta_{high}$ to mitigate false merges. We provide more details in Appendix B.

In Section 5, we mention an optimization for the binary signatures, based on blocking, which empirically improves the approximation quality, while reducing memory and computational overhead.

4.2 Convergence and Hash Analysis

The running time of our algorithm depends primarily on the number of iterations and the total number of comparisons performed. The two types of comparisons are edit distance computations, which take time $O(rm)$ to check distance at most r, and q-gram distance computations, which take time linear in the signature length. To avoid unnecessary comparisons, we partition cluster representatives using our hash function and only compare reads with the same hash value. Therefore, we bound the total number of comparisons by bounding the total number of hash collisions. In particular, we prove the following convergence theorem (details appear in Appendix C).

Theorem 4.1 (Informal). *For sufficiently large n and m and small p, there exist parameters for our algorithm such that it outputs a clustering with accuracy $(1-\varepsilon)$ and the expected number of comparisons is*

$$O\left(\max\left\{n^{1+O(p)},\ \frac{n^2}{m^{\Omega(1/p)}}\right\} \cdot \left(1 + \frac{\log(s/\varepsilon)}{s}\right)\right).$$

Note that $n^{1+O(p)} \geqslant n^2/m^{\Omega(1/p)}$ in the expression above whenever the reads are long enough, that is, when $m \geqslant n^{cp}$ (where c is some small constant). Thus, for a large range of n, m, p, and ε, our algorithm converges in time proportional to $n^{1+O(p)}$, which is sub-quadratic in n, the number of input reads. Since we expect the number of clusters k to be $k = \Omega(n)$, our algorithm outperforms any methods that require time $\Omega(kn) = \Omega(n^2)$ in this regime.

The running time analysis of our algorithm revolves around estimating both the collision probability of our hash function and the overall convergence time to identify the underlying clusters. The main overhead comes from unnecessarily comparing reads that belong to different clusters. Indeed, for pairs of reads inside the same cluster, the total number of comparisons is $O(n)$, since after a comparison, the reads will merge into the same cluster. For reads in different clusters, we show that they collide with probability that is exponentially small in the hash length (since they are nearly-random strings). For the convergence analysis, we prove that reads in the same cluster will collide with significant probability, implying that after roughly

$$O\left(\max\left\{n^{O(p)},\ \frac{n}{m^{\Omega(1/p)}}\right\} \cdot \left(1 + \frac{\log(s/\varepsilon)}{s}\right)\right)$$

iterations, the found clustering will be $(1-\varepsilon)$ accurate.

In Section 5, we experimentally validate our algorithm's running time, convergence, and correctness properties on real and synthetic data.

4.3 Outlier Robustness

Our final theoretical result involves bounding the number of incorrect merges caused by potential outliers in the dataset. In real datasets, we expect some number of highly-noisy reads, due to experimental error. Fortunately, such outliers lead to only a minor loss in accuracy for our algorithm, when the clusters are separated. We prove the following theorem in Appendix D.

Theorem 4.2. *Let $\mathbf{C} = \{C_1, \ldots, C_k\}$ be an $(r, 2r)$-separated clustering of S. Let O be any set of size $\varepsilon' k$. Fixing the randomness and parameters in the algorithm with distance threshold r, let $\widetilde{\mathbf{C}}$ be the output on S and $\tilde{\mathbf{C}}'$ be the output on $S \cup \mathsf{O}$. Then, $\mathcal{A}_\gamma(\mathbf{C}, \tilde{\mathbf{C}}') \geqslant \mathcal{A}_\gamma(\mathbf{C}, \tilde{\mathbf{C}}) - \varepsilon'$.*

Notice that this is optimal since $\varepsilon' k$ outliers can clearly modify $\varepsilon' k$ clusters. For DNA storage data recovery, if we desire $1 - \varepsilon$ accuracy overall, and we expect at most $\varepsilon' k$ outliers, then we simply need to aim for a clustering with accuracy at least $1 - \varepsilon + \varepsilon'$.

5 Experiments

We experimentally evaluate our algorithm on real and synthetic data, measuring accuracy and wall clock time. Table 1 describes our datasets. We evaluate accuracy on the real data by comparing the found clusterings to a gold standard clustering. We construct the gold standard by using the original reference strands, and we group the reads by their most likely reference using an established alignment tool (see Appendix E for full details). The synthetically generated data resembles real data distributions and properties [45]. We implement our algorithm in C++ using MPI. We run tests on Microsoft Azure virtual machines (size H16mr: 16 cores, 224 GB RAM, RDMA network).

Table 1: Datasets. Real data from Organick et. al. [45]. Synthetic data from Defn. 2.3. Appendix E has details.

Dataset	# Reads	Avg. Length	Description
3.1M real	3,103,511	150	Movie file stored in DNA
13.2M real	13,256,431	150	Music file stored in DNA
58M real	58,292,299	150	Collection of files (40MB stored in DNA; includes above)
12M real	11,973,538	110	Text file stored in DNA
5.3B synthetic	5,368,709,120	110	Noise $p = 4\%$; cluster size $s = 10$.

5.1 Implementation and Parameter Details

For the edit distance threshold, we desire r to be just larger than the cluster diameter. With p noise, we expect the diameter to be at most $4pm$ with high probability. We conservatively estimate $p \approx 4\%$ for real data, and thus we set $r = 25$, since $4pm = 24$ for $p = 0.04$ and $m = 150$.

For the binary signatures, we observe that choosing larger q separates clusters better, but it also increases overhead, since $\sigma_q(x) \in \{0,1\}^{4^q}$ is very high-dimensional. To remedy this, we used a blocking approach. We partitioned x into blocks of 22 characters and computed σ_3 of each block, concatenating these 64-bit strings for the final signature. On synthetic data, we found that setting $\theta_{low} = 40$ and $\theta_{high} = 60$ leads to very reduced running time while sacrificing negligible accuracy.

For the hashing, we set w, ℓ to encourage collisions of close pairs and discourage collisions of far pairs. Following Theorem C.1, we set $w = \lceil \log_4(m) \rceil = 4$ and $\ell = 12$, so that $w + \ell = 16 = \log_4 n$ with $n = 2^{32}$. Since our clusters are very small, we find that we can further filter far pairs by concatenating two independent hashes to define a bucket based on this 64-bit value. Moreover, since we expect very few reads to have the same hash, instead of comparing all pairs in a hash bucket, we sort the reads based on hash value and only compare adjacent elements. For communication, we use only the first 20 bits of the hash value, and we uniformly distribute clusters based on this.

Finally, we conservatively set the number of iterations to 780 total (26 communication rounds, each with 30 local iterations) because this led to 99.9% accuracy on synthetic data (even with $\gamma = 1.0$).

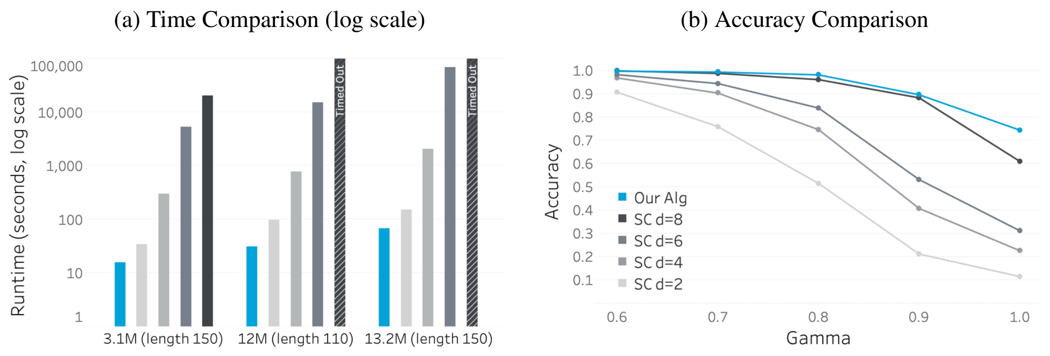

Figure 2: Comparison to Starcode. Figure 2a plots running times on three real datasets of our algorithm versus four Starcode executions using four distance thresholds $d \in \{2, 4, 6, 8\}$. For the first dataset, with 3.1M real reads, Figure 2b plots $\mathcal{A}_\gamma$ for varying $\gamma \in \{0.6, 0.7, 0.8, 0.9, 1.0\}$ of our algorithm versus Starcode. We stopped Starcode if it did not finish within 28 hours. We ran tests on one processor, 16 threads.

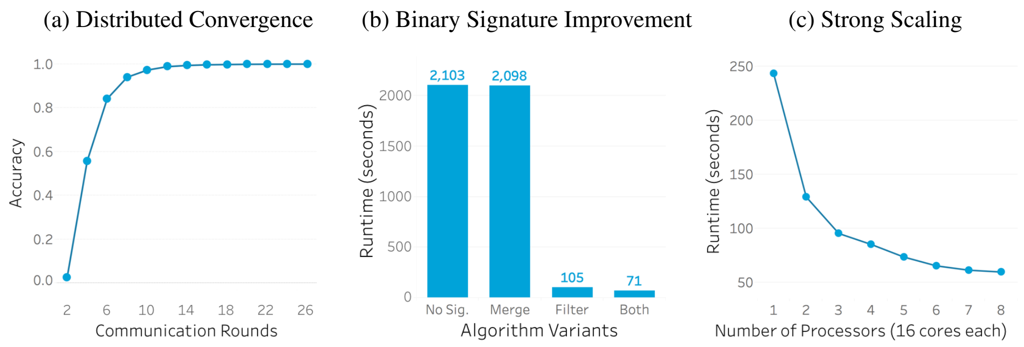

Figure 3: Empirical results for our algorithm. Figure 3a plots accuracy $\mathcal{A}_{0.9}$ of intermediate clusterings (5.3B synthetic reads, 24 processors). Figure 3b shows single-threaded running times for four variants of our algorithm, depending on whether it uses signatures for merging and/or filtering (3.1M real reads; single thread). Figure 3c plots times as the number of processors varies from 1 to 8, with 16 cores per processor (58M real reads).

Starcode Parameters Starcode [57] takes a distance threshold $d \in \{1, 2, \ldots, 8\}$ as an input parameter and finds all clusters with radius not exceeding this threshold. We run Starcode for various settings of d, with the intention of understanding how Starcode's accuracy and running time change with this parameter. We use Starcode's sphere clustering "-s" option, since this has performed most accurately on sample data, and we use the "-t" parameter to run Starcode with 16 threads.

5.2 Discussion

Figure 2 shows that our algorithm outperforms Starcode, the state-of-the-art clustering algorithm for DNA sequences [57], in both accuracy and time. As explained above, we have set our algorithm's parameters based on theoretical estimates. On the other hand, we vary Starcode's distance threshold parameter $d \in \{2, 4, 6, 8\}$. We demonstrate in Figures 2a and 2b that increasing this distance parameter significantly improves accuracy on real data, but also it also greatly increases Starcode's running time. Both algorithms achieve high accuracy for $\gamma = 0.6$, and the gap between the algorithms widens as γ increases. In Figure 2a, we show that our algorithm achieves more than a 1000x speedup over the most accurate setting of Starcode on three real datasets of varying sizes and read lengths. For $d \in \{2, 4, 6\}$, our algorithm has a smaller speedup and a larger improvement in accuracy.

Figure 3a shows how our algorithm's clustering accuracy increases with the number of communication rounds, where we evaluate $\mathcal{A}_\gamma$ with $\gamma = 0.9$. Clearly, using 26 rounds is quite conservative. Nonetheless, our algorithm took only 46 minutes wall clock time to cluster 5.3B synthetic reads on 24 processors (384 cores). We remark that distributed MapReduce-based algorithms for string similarity joins have been reported to need tens of minutes for only tens of millions of reads [21, 51].

Figure 3b demonstrates the effect of binary signatures on runtime. Recall that our algorithm uses signatures in two places: merging clusters when $d_H(\sigma(x), \sigma(y)) \leqslant \theta_{low}$, and filtering pairs when $d_H(\sigma(x), \sigma(y)) > \theta_{high}$. This leads to four natural variants: (i) omitting signatures, (ii) using them for merging, (iii) using them for filtering, or (iv) both. The biggest improvement (20x speedup) comes from using signatures for filtering (comparing (i) vs. (iii)). This occurs because the cheap Hamming distance filter avoids a large number of expensive edit distance computations. Using signatures for merging provides a modest 30% improvement (comparing (iii) vs. (iv)); this gain does not appear between (i) and (ii) because of time it takes to compute the signatures. Overall, the effectiveness of signatures justifies their incorporation into an algorithm that already filters based on hashing.

Figure 3c evaluates the scalability of our algorithm on 58M real reads as the number of processors varies from 1 to 8. At first, more processors lead to almost optimal speedups. Then, the communication overhead outweighs the parallelization gain. Achieving perfect scalability requires greater understanding and control of the underlying hardware and is left as future work.

6 Related Work

Recent work identifies the difficulty of clustering datasets containing large numbers of small clusters. Betancourt et. al. [11] calls this “microclustering” and proposes a Bayesian non-parametric model for entity resolution datasets. Kobren et. al. [37] calls this “extreme clustering” and studies hierarchical clustering methods. DNA data storage provides a new domain for micro/extreme clustering, with interesting datasets and important consequences [12, 24, 26, 45, 52].

Large-scale, extreme datasets – with billions of elements and hundreds of millions of clusters – are an obstacle for many clustering techniques [19, 29, 33, 42]. We demonstrate that DNA datasets are well-separated, which implies that our algorithm converges quickly to a highly-accurate solution. It would be interesting to determine the minimum requirements for robustness in extreme clustering.

One challenge of clustering for DNA storage comes from the fact that reads are strings with edit errors and a four-character alphabet. Edit distance is regarded as a difficult metric, with known lower bounds in various models [1, 5, 7]. Similarity search algorithms based on MinHash [13, 14] originally aimed to find duplicate webpages or search results, which have much larger natural language alphabets. However, known MinHash optimizations [40, 41] may improve our clustering algorithm.

Chakraborty, Goldenberg, and Koucký explore the question of preserving small edit distances with a binary embedding [16]. This embedding was adapted by Zhang and Zhang [56] for approximate string similarity joins. We leave a thorough comparison to these papers as future work, along with obtaining better theoretical bounds for hashing or embeddings [17, 46] under our data distribution.

7 Conclusion

We highlighted a clustering task motivated by DNA data storage. We proposed a new distributed algorithm and hashing scheme for edit distance. Experimentally and theoretically, we demonstrated our algorithm’s effectiveness in terms of accuracy, performance, scalability, and robustness.

We plan to release one of our real datasets. We hope our dataset and data model will lead to further research on clustering and similarity search for computational biology or other domains with strings.

For future work, our techniques may also apply to other metrics and to other applications with large numbers of small, well-separated clusters, such as entity resolution or deduplication [20, 23, 32]. Finally, our work motivates a variety of new theoretical questions, such as studying the distortion of embeddings for random strings under our generative model (we elaborate on this in Appendix B).

8 Acknowledgments

We thank Yair Bartal, Phil Bernstein, Nova Fandina, Abe Friesen, Sariel Har-Peled, Christian Konig, Paris Koutris, Marina Meila, Mark Yatskar for useful discussions. We also thank Alyshia Olsen for help designing the graphs. Finally, we thank Jacob Nelson for sharing his MPI wisdom and Taylor Newill and Christian Smith from the Microsoft Azure HPC Team for help using MPI on Azure.

References

[1] A. Abboud, T. D. Hansen, V. V. Williams, and R. Williams. Simulating Branching Programs with Edit Distance and Friends: Or: A polylog shaved is a lower bound made. In *STOC*, 2016.

[2] M. Ackerman, S. Ben-David, D. Loker, and S. Sabato. Clustering Oligarchies. In *AISTATS*, 2013.

[3] M. Ackerman and S. Dasgupta. Incremental Clustering: The Case for Extra Clusters. In *Advances in Neural Information Processing Systems*, pages 307–315, 2014.

[4] N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: ranking and clustering. *Journal of the ACM (JACM)*, 55(5):23, 2008.

[5] A. Andoni and R. Krauthgamer. The Computational Hardness of Estimating Edit Distance. *SIAM J. Comput.*, 39(6).

[6] A. Andoni and R. Krauthgamer. The Smoothed Complexity of Edit Distance. *ACM Transactions on Algorithms (TALG)*, 8(4):44, 2012.

[7] A. Backurs and P. Indyk. Edit Distance Cannot be Computed in Strongly Subquadratic time (unless SETH is false). In *STOC*, 2015.

[8] M.-F. Balcan, Y. Liang, and P. Gupta. Robust Hierarchical Clustering. *Journal of Machine Learning Research*, 15(1):3831–3871, 2014.

[9] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Machine Learning*, 56(1-3):89–113, 2004.

[10] T. Batu, F. Ergün, J. Kilian, A. Magen, S. Raskhodnikova, R. Rubinfeld, and R. Sami. A Sublinear Algorithm for Weakly Approximating Edit Distance. In *STOC*, 2003.

[11] B. Betancourt, G. Zanella, J. W. Miller, H. Wallach, A. Zaidi, and B. Steorts. Flexible Models for Microclustering with Application to Entity Resolution. In *NIPS*, 2016.

[12] J. Bornholt, R. Lopez, D. M. Carmean, L. Ceze, G. Seelig, and K. Strauss. A DNA-based Archival Storage System. In *ASPLOS*, 2016.

[13] A. Z. Broder. On the Resemblance and Containment of Documents. In *Compression and Complexity of Sequences*, pages 21–29. IEEE, 1997.

[14] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic Clustering of the Web. *Computer Networks and ISDN Systems*, 29(8-13):1157–1166, 1997.

[15] J. Buhler. Efficient large-scale sequence comparison by locality-sensitive hashing. *Bioinformatics*, 17(5):419–428, 2001.

[16] D. Chakraborty, E. Goldenberg, and M. Koucký. Streaming Algorithms for Embedding and Computing Edit Distance in the Low Distance Regime. In *STOC*, 2016.

[17] M. Charikar and R. Krauthgamer. Embedding the Ulam Metric into L_1. *Theory of Computing*, 2(11):207–224, 2006.

[18] S. Chawla, K. Makarychev, T. Schramm, and G. Yaroslavtsev. Near Optimal LP Rounding Algorithm for Correlation Clustering on Complete and Complete k-partite Graphs. In *STOC*, 2015.

[19] J. Chen, H. Sun, D. Woodruff, and Q. Zhang. Communication-Optimal Distributed Clustering. In *Advances in Neural Information Processing Systems*, pages 3720–3728, 2016.

[20] P. Christen. *Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection*. Springer Science & Business Media, 2012.

[21] D. Deng, G. Li, S. Hao, J. Wang, and J. Feng. Massjoin: A Mapreduce-based Method for Scalable String Similarity Joins. In *ICDE*, pages 340–351. IEEE, 2014.

[22] D. P. Dubhashi and A. Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009.

[23] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Transactions on knowledge and data engineering*, 19(1):1–16, 2007.

[24] Y. Erlich and D. Zielinski. DNA Fountain Enables a Robust and Efficient Storage Architecture. *Science*, 355(6328):950–954, 2017.

[25] S. Ganguly, E. Mossel, and M. Z. Rácz. Sequence Assembly from Corrupted Shotgun Reads. In *ISIT*, pages 265–269, 2016. `http://arxiv.org/abs/1601.07086`.

[26] 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*, 494(7435), 2013.

[27] S. Gollapudi and R. Panigrahy. A Dictionary for Approximate String Search and Longest Prefix Search. In *CIKM*, 2006.

[28] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, D. Srivastava, et al. Approximate String Joins in a Database (almost) for Free. In *VLDB*, volume 1, pages 491–500, 2001.

[29] S. Guha, Y. Li, and Q. Zhang. Distributed Partial Clustering. *arXiv preprint arXiv:1703.01539*, 2017.

[30] H. Hanada, M. Kudo, and A. Nakamura. On Practical Accuracy of Edit Distance Approximation Algorithms. *arXiv preprint arXiv:1701.06134*, 2017.

[31] S. Har-Peled, P. Indyk, and R. Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory of Computing*, 8(1):321–350, 2012.

[32] O. Hassanzadeh, F. Chiang, H. C. Lee, and R. J. Miller. Framework for Evaluating Clustering Algorithms in Duplicate Detection. *PVLDB*, 2(1):1282–1293, 2009.

[33] C. Hennig, M. Meila, F. Murtagh, and R. Rocci. *Handbook of Cluster Analysis*. CRC Press, 2015.

[34] Y. Jiang, D. Deng, J. Wang, G. Li, and J. Feng. Efficient Parallel Partition-based Algorithms for Similarity Search and Join with Edit Distance Constraints. In *Joint EDBT/ICDT Workshops*, 2013.

[35] Y. Jiang, G. Li, J. Feng, and W.-S. Li. String Similarity Joins: An Experimental Evaluation. *PVLDB*, 7(8):625–636, 2014.

[36] J. Johnson, M. Douze, and H. Jégou. Billion-scale Similarity Search with GPUs. *arXiv preprint arXiv:1702.08734*, 2017.

[37] A. Kobren, N. Monath, A. Krishnamurthy, and A. McCallum. A Hierarchical Algorithm for Extreme Clustering. In *KDD*, 2017.

[38] R. Krauthgamer and Y. Rabani. Improved Lower Bounds for Embeddings Into L_1. *SIAM J. on Computing*, 38(6):2487–2498, 2009.

[39] H. Li and R. Durbin. Fast and Accurate Short Read Alignment with Burrows–Wheeler Transform. *Bioinformatics*, 25(14):1754–1760, 2009.

[40] P. Li and C. König. b-Bit Minwise Hashing. In *WWW*, pages 671–680. ACM, 2010.

[41] P. Li, A. Owen, and C.-H. Zhang. One Permutation Hashing. In *NIPS*, 2012.

[42] G. Malkomes, M. J. Kusner, W. Chen, K. Q. Weinberger, and B. Moseley. Fast Distributed k-center Clustering with Outliers on Massive Data. In *NIPS*, 2015.

[43] J. Matoušek. *Lectures on Discrete Geometry*, volume 212. Springer New York, 2002.

[44] M. Meilă and D. Heckerman. An Experimental Comparison of Model-based Clustering Methods. *Machine learning*, 42(1-2):9–29, 2001.

[45] L. Organick, S. D. Ang, Y.-J. Chen, R. Lopez, S. Yekhanin, K. Makarychev, M. Z. Racz, G. Kamath, P. Gopalan, B. Nguyen, C. Takahashi, S. Newman, H.-Y. Parker, C. Rashtchian, K. Stewart, G. Gupta, R. Carlson, J. Mulligan, D. Carmean, G. Seelig, L. Ceze, and K. Strauss. Scaling up DNA data storage and random access retrieval. *bioRxiv*, 2017.

[46] R. Ostrovsky and Y. Rabani. Low Distortion Embeddings for Edit Distance. *J. ACM*, 2007.

[47] X. Pan, D. Papailiopoulos, S. Oymak, B. Recht, K. Ramchandran, and M. I. Jordan. Parallel Correlation Clustering on Big Graphs. In *Advances in Neural Information Processing Systems*, pages 82–90, 2015.

[48] Z. Rasheed, H. Rangwala, and D. Barbara. Efficient Clustering of Metagenomic Sequences using Locality Sensitive Hashing. In *Proceedings of the 2012 SIAM International Conference on Data Mining*, pages 1023–1034. SIAM, 2012.

[49] N. Sundaram, A. Turmukhametova, N. Satish, T. Mostak, P. Indyk, S. Madden, and P. Dubey. Streaming Similarity Search Over One Billion Tweets Using Parallel Locality-Sensitive Hashing. *PVLDB*, 6(14):1930–1941, 2013.

[50] E. Ukkonen. Approximate String-matching with q-grams and Maximal Matches. *Theoretical computer science*, 92(1):191–211, 1992.

[51] C. Yan, X. Zhao, Q. Zhang, and Y. Huang. Efficient string similarity join in multi-core and distributed systems. *PloS one*, 12(3):e0172526, 2017.

[52] S. H. T. Yazdi, R. Gabrys, and O. Milenkovic. Portable and Error-Free DNA-Based Data Storage. *bioRxiv*, page 079442, 2016.

[53] M. Yu, G. Li, D. Deng, and J. Feng. String Similarity Search and Join: A Survey. *Frontiers of Computer Science*, 10(3):399–417, 2016.

[54] P. Yuan, C. Sha, and Y. Sun. Hashˆ{ed}-Join: Approximate String Similarity Join with Hashing. In *International Conference on Database Systems for Advanced Applications*, pages 217–229. Springer, 2014.

[55] R. B. Zadeh and A. Goel. Dimension Independent Similarity Computation. *The Journal of Machine Learning Research*, 14(1):1605–1626, 2013.

[56] H. Zhang and Q. Zhang. EmbedJoin: Efficient Edit Similarity Joins via Embeddings. In *KDD*, 2017.

[57] E. V. Zorita, P. Cuscó, and G. Filion. Starcode: Sequence Clustering Based on All-pairs Search. *Bioinformatics*, 2015.

A Clusters are Well-Separated in Edit Distance

Let $\mathcal{U}_m$ be the uniform distribution over Σ^m. Recall that in our data model the cluster centers $c_1, \ldots, c_k$ are independently sampled from $\mathcal{U}_m$. Therefore, in this section, we analyze the edit distance between random strings. We also bound the cluster radius when clusters consist of p-noisy copies of the cluster centers (Definition 2.2). We will prove that a clustering $\mathbf{C} = \{C_1, \ldots, C_k\}$ is well-separated when each C_i is sampled from $\mathcal{D}_{s,p,m}$ for $i \in [k]$ where we define $[k] = \{1, 2, \ldots, k\}$. The separation will hold even if the error rate is constant (that is, when p is a small constant).

We need the following concentration bound for our analyses.

Lemma A.1 ([22]). *Let f be a function of m random variables $Z_1, \ldots, Z_m$ such that $f(Z_1, \ldots, Z_m) \leqslant b$ for some b. For $i \in [m]$ let ξ_i satisfy*

$$|\mathbb{E}[f \mid Z_1, \ldots, Z_{i-1}, Z_i = a_i] - \mathbb{E}[f \mid Z_1, \ldots, Z_{i-1}, Z_i = a'_i]| \leqslant \xi_i.$$

Then for any $\eta > 0$, we have that

$$\Pr\left[|f - \mathbb{E}[f]| > \eta\right] \leqslant \exp\left(-\frac{2\eta^2}{\sum_{i=1}^m \xi_i^2}\right)$$

A.1 Edit Distance of Random Strings

There exists a constant $c_{\mathsf{ind}} = c_{\mathsf{ind}}(|\Sigma|) > 0$ such that for $x, y \sim \mathcal{U}_m$,

$$\lim_{m\to\infty} \frac{d_E(x,y)}{m} = c_{\text{ind}}$$

almost surely (this follows from Kingman's ergodic theorem). Determining the precise value of c_{ind} is a challenging open problem, related to calculating the size of a ball under edit distance [10, 25]. When $|\Sigma| = 4$, simulations suggest $c_{\text{ind}} \approx 0.51$, and it is known [25] that $c_{\text{ind}} > 0.338$. In what follows, we will use that c_{ind} is the largest constant satisfying

$$\mathop{\mathbb{E}}_{x,y\sim\mathcal{U}_m}\left[d_E(x,y)\right] \geqslant c_{\mathsf{ind}} \cdot m,$$

for $\Sigma = \{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}\}$.

Lemma A.2. *For any $\lambda > 0$ we have*

$$\Pr_{x,y\sim\mathcal{U}_m}\left[\left|d_E(x,y) - \mathop{\mathbb{E}}_{x,y\sim\mathcal{U}_m}\left[d_E(x,y)\right]\right| \leqslant \lambda\sqrt{\frac{m}{2}}\,\right] \geqslant 1 - e^{-\lambda^2}.$$

Proof. Let $X = X_1X_2\cdots X_m$ and $Y = Y_1Y_2\cdots Y_m$ be random strings drawn from $\mathcal{U}_m$, and let Z_i be the pair (X_i, Y_i). Define $f(Z_1, \ldots, Z_m) = d_E(X, Y)$. It is easy to see that

$$|\mathbb{E}[f \mid Z_1, \ldots, Z_{i-1}, Z_i = (x_i, y_i)] - \mathbb{E}[f \mid Z_1, \ldots, Z_{i-1}, Z_i = (x'_i, y'_i)]| \leqslant 1,$$

where x_i, y_i, x'_i, y'_i are any four characters in Σ. Thus, the statement follows from Lemma A.1. □

The above lemma implies that random strings will have edit distance that is linear in m with high probability, since their expected distance is at least $c_{\mathsf{ind}} \cdot m$ and at most m.

A.2 Cluster Analysis for Edit Distance

We first analyze a cluster's radius by bounding the edit distance of p-noisy copies.

Lemma A.3. *Let x be a p-noisy copy of $c \in \Sigma^m$. Then, for any $\lambda > 0$ we have*

$$\Pr\left[d_E(x,c) \leqslant pm + \lambda\sqrt{3m}\,\right] \geqslant 1 - e^{-\lambda^2}.$$

Proof. Recall that x was generated from c by going from left to right in c, and at each character, introducing an edit (substitution, insertion, or deletion) with probability p, independently of the other edits. Notice that the number of edits from c to x is binomially distributed, with parameters m and p. Therefore, $\mathbb{E}\left[d_E(x,c)\right] = pm$, and a standard Chernoff bound proves the lemma. □

We tie together the above lemmas to prove that clusters are separated with high probability under edit distance when m is large enough. In what follows, we think of p as a small constant $p \ll c_{\mathsf{ind}}$, to ensure separation. For high probability bounds, the number of input reads n just needs to satisfy

$$n < e^{\delta' c_{\mathsf{ind}}^2 m}$$

for a small enough constant $\delta' > 0$.

Lemma A.4. *Consider a clustering* $\mathbf{C} = \{C_1, \ldots, C_k\}$*, where each* $C_i \sim \mathcal{D}_{s,p,m}$*, and let* $n = sk$*. Then* $\mathbf{C}$ *is* (r_1, r_2)*-separated for*

$$r_1 = 2pm + 3\sqrt{m \ln n}$$
$$r_2 = c_{\mathsf{ind}} \cdot m - 4pm - 12\sqrt{m \ln n},$$

with probability at least $1 - 1/n^2$.

Proof. Let $\mathcal{E}_1$ be the event that for all $i \in [k]$ and for all $x \in C_i$, we have

$$d_E(x, c_i) \leqslant \frac{r_1}{2},$$

where c_i is the center of C_i. We claim that $\mathcal{E}_1$ holds with probability at least $1 - \frac{1}{2n^2}$. This follows from setting $\lambda = 2\sqrt{\ln n}$ in Lemma A.3 and a union bound over the $sk = n$ pairs (x, c_i) relevant to the event $\mathcal{E}_1$. Notice that when $\mathcal{E}_1$ holds, we have that $d_E(x, x') \leqslant r_1$ for any pair $x, x' \in C_i$.

Now, recall that

$$\mathbb{E}\left[d_E(c_i, c_j)\right] \geqslant c_{\mathsf{ind}} \cdot m,$$

and let $\mathcal{E}_2$ be the event that for all pairs of cluster centers c_i and c_j for $i \neq j$ we have

$$d_E(c_i, c_j) > r_2 + 2r_1.$$

When $\mathcal{E}_1 \cap \mathcal{E}_2$ holds, we have that for any two points $x \in C_i$ and $y \in C_j$ with $i \neq j$,

$$d_E(x, y) \geqslant d_E(c_i, c_j) - d_E(x, c_i) - d_E(y, c_j) > r_2 + 2r_1 - 2r_1 = r_2,$$

where the first inequality follows from the triangle inequality. By setting $\lambda = 2\sqrt{\ln n}$ in Lemma A.2 and a union bound over $\binom{k}{2} \leqslant n^2$ pairs of cluster centers, we have that $\mathcal{E}_2$ holds with probability at least $1 - \frac{1}{2n^2}$.

We conclude that $\mathcal{E}_1 \cap \mathcal{E}_2$ holds with probability at least $1 - \frac{1}{n^2}$ by a union bound, and that $\mathbf{C}$ is (r_1, r_2)-separated conditioned on $\mathcal{E}_1 \cap \mathcal{E}_2$. □

B Clusters are Well-Separated Under Binary Signatures

Analogously to the previous section, we now analyze the q-gram distance, that is, the Hamming distance between binary signatures. We also analyze the cluster separation for random centers.

B.1 Signature Distance of Random Strings

We start by lower bounding the expected distance for random strings.

Lemma B.1. *Let* q *be a natural number satisfying* $4^q \geqslant 2m$ *and* $m \geqslant 2q$*. Then,*

$$\mathop{\mathbb{E}}_{x,y \sim \mathcal{U}_m}\left[d_H(\sigma_q(x), \sigma_q(y))\right] \geqslant 2(m - q + 1) - \frac{2(m - q + 1)^2}{4^q}.$$

In particular, for $q = \lceil 2 \log_4 m \rceil$ *and* m *large enough,* $d_H(\sigma_q(c_i), \sigma_q(c_i)) \geqslant 2m - O(\log m)$ *in expectation for distinct cluster centers* c_i, c_j.

Proof. The number of q-grams in a string $x \in \Sigma^m$ is exactly $m - q + 1$. Let X, Y denote the multi-set of q-grams in x and y, respectively. Observe that

$$d_H(\sigma_q(x), \sigma_q(y)) = 2(m - q + 1) - 2|X \cap Y|.$$

By linearity of expectation, we simply need to prove that $\mathbb{E}[|X \cap Y|] \leqslant |X||Y|/4^q$. For each element of X, the probability it is contained in Y is at most $|Y|/4^q$, since there are 4^q possible q-grams. Therefore the bound follows by summing over the $|X|$ q-grams in X. For distinct centers c_i and c_j, the lower bound on the expectation of $d_H(\sigma_q(c_i), \sigma_q(c_i))$ follows by plugging in $q = \lceil 2 \log_4 m \rceil$. □

Lemma B.2. *Let $\mathcal{U}_m$ be the uniform distribution over Σ^m. Let q be a natural number satisfying $4^q \geqslant 2m$ and $m \geqslant 2q$. Then, for any $\lambda > 0$, we have*

$$\Pr_{x,y\sim\mathcal{U}_m}\left[\left|d_H(\sigma_q(x),\sigma_q(y)) - \mathbb{E}[d_H(\sigma_q(x),\sigma_q(y))]\right| \leqslant \lambda q\sqrt{m}\right] \geqslant 1 - e^{-2\lambda^2}.$$

Proof. Let $X = X_1X_2\cdots X_m$ and $Y = Y_1Y_2\cdots Y_m$ be random strings drawn from $\mathcal{U}_m$, and let Z_i be the pair (X_i, Y_i). Define $f(Z_1,\ldots,Z_m) = d_H(\sigma_q(X),\sigma_q(Y))$. Since σ_q is defined in terms of q-grams, it is easy to see that

$$|\mathbb{E}[f \mid Z_1,\ldots,Z_{i-1}, Z_i = (x_i,y_i)] - \mathbb{E}[f \mid Z_1,\ldots,Z_{i-1}, Z_i = (x_i',y_i')]| \leqslant q,$$

where x_i, y_i, x_i', y_i' are any four characters in Σ^m. Therefore, the statement follows from Lemma A.1. □

B.2 Cluster Analysis for Signatures

We are interested in bounding the maximum difference between the edit distance of the original strands and the Hamming distance of the signatures.

Lemma B.3. *Let q be a natural number satisfying $0 \leqslant q \leqslant m$. For any $x, y \in \Sigma^*$,*

$$d_H(\sigma_q(x),\sigma_q(y)) \leqslant \min\{2q\cdot d_E(x,y),\ |x| + |y| - 2q + 2\}$$

Proof. The upper bound of $|x|+|y|-2q+2$ holds simply because $\sigma_q(x)$ contains at most $|x|-q+1$ non-zero coordinates (likewise for y). To show $d_H(\sigma_q(x),\sigma_q(y)) \leqslant 2q\cdot d_E(x,y)$, we will analyze the case of a single edit, and prove $d_H(\sigma_q(x),\sigma_q(z)) \leqslant 2q$ when $d_E(x,z) = 1$. Then, we observe that the triangle inequality implies the upper bound for all edit distances. Let z be any string in Σ^* with $d_E(x,z) = 1$. Let X and Z be the set of q-grams in x and z, respectively. Notice that $d_H(\sigma_q(x),\sigma_q(z))$ equals the size of the symmetric difference $|X\triangle Z|$. We claim $|X \setminus Z| \leqslant q$, since the single edit between x and z can cause at most q elements of X to be absent in Z. Similarly, we claim $|Z\setminus X| \leqslant q$, since the single edit between x and z can cause at most q elements to be in Z that are not present in X. Thus,

$$d_H(\sigma_q(x),\sigma_q(z)) = |X\triangle Z| \leqslant |X\setminus Z| + |Z\setminus X| \leqslant 2q,$$

as desired. □

Lemma B.4. *Let x be a p-noisy copy of $c \in \Sigma^m$. Then, for any $\lambda > 0$ we have*

$$\Pr\left[d_H(\sigma_q(x),\sigma_q(c)) \leqslant 2q\left(pm + \lambda\sqrt{3m}\right)\ \right] \geqslant 1 - e^{-\lambda^2}.$$

Proof. This follows directly from Lemmas A.3 and B.3. □

Lemma B.5. *Let $q = \lceil \log_4 m\rceil$. Let $\mathbf{C} = \{C_1,\ldots,C_k\}$ be a random clustering with $C_i \sim \mathcal{D}_{s,p,m}$. Then, with probability $1 - 1/n^2$, we have that*

1. *for any $i = 1,2,\ldots,k$ and any $x,y \in C_i$,*

$$d_H(\sigma_q(x),\sigma_q(y)) \leqslant 4q\left(pm + 3\sqrt{m\ln n}\right) =: r_1'.$$

2. *for $x \in C_i$ and $y \in C_j$ with $i \neq j$,*

$$d_H(\sigma_q(x),\sigma_q(y)) \geqslant 2m - 2q\left(4pm + 7\sqrt{m\ln n}\right) =: r_2'.$$

Proof. Let $\mathcal{E}_1'$ be the event that for all $i \in [k]$ and for all $x \in C_i$, we have

$$d_H(\sigma_q(x),\sigma_q(c_i)) \leqslant \frac{r_1'}{2},$$

where c_i is the center of C_i. We claim that $\mathcal{E}_1'$ holds with probability at least $1 - \frac{1}{2n^2}$. This follows from setting $\lambda = 2\sqrt{\ln n}$ in Lemma B.4 and a union bound over the $sk = n$ pairs (x, c_i) relevant to the event $\mathcal{E}_1'$. Notice that when $\mathcal{E}_1'$ holds, we have that $d_H(\sigma_q(x), \sigma_q(x')) \leqslant r_1'$ for any pair $x, x' \in C_i$.

Now, let $\mathcal{E}_2'$ be the event that for all pairs of cluster centers c_i and c_j for $i \neq j$ we have

$$d_H(\sigma_q(c_i), \sigma_q(c_j)) > r_2' + 2r_1'.$$

Notice that when $\mathcal{E}_1' \cap \mathcal{E}_2'$ holds, we have that for any two points $x \in C_i$ and $y \in C_j$ with $i \neq j$,

$$d_H(\sigma_q(x), \sigma_q(y))) \geqslant d_H(\sigma_q(c_i), \sigma_q(c_j)) - d_H(\sigma_q(x), \sigma_q(c_i)) - d_H(\sigma_q(y), \sigma_q(c_j)) > r_2',$$

where the first inequality follows from the triangle inequality. By setting $\lambda = 2\sqrt{\ln n}$ in Lemma B.2 and a union bound over $\binom{k}{2} \leqslant n^2$ pairs of cluster centers, we have that $\mathcal{E}_2'$ holds with probability at least $1 - \frac{1}{2n^2}$.

We conclude that $\mathcal{E}_1' \cap \mathcal{E}_2'$ holds with probability at least $1 - \frac{1}{n^2}$ by a union bound, and that $\mathbf{C}$ satisfies the claimed bounds conditioned on $\mathcal{E}_1' \cap \mathcal{E}_2'$. $\square$

Lemma B.5 proves that the clusters are separated according to the binary signatures with high probability whenever $p \ll 1/\log m$. This is a stricter requirement than we needed for the edit distance separation, since that tolerated error rate $p = O(1)$. Constructing an efficient binary embedding for $p = \Omega(1)$ would indeed be interesting.

B.3 Metric Embedding Related Work

We briefly mention a connection to metric embeddings for non-repetitive strings. Charikar and Krauthgamer [17] call $x \in \Sigma^m$ a *t-non-repetitive* string if all of its t-grams are distinct. They provide an embedding into the ℓ_1 metric space with distortion $O(t \log m)$ for the submetric of edit distance corresponding to considering only t-non-repetitive stings. For random strings, we prove in Lemmas B.2 and B.3 that the binary signatures $\sigma_q(x)$ provide an embedding into Hamming space with distortion $O(\log m)$ with high probability when $q = 2\log_4 m$. It is natural to wonder if our binary signatures work in general for $O(\log m)$-non-repetitive strings (since random strings will have this property with high probability). Unfortunately, it is easy to construct pairs of example strings with linear edit distance but whose signatures have only logarithmic Hamming distance ($\approx q$). Therefore, it is an interesting open question to find an efficient metric embedding into ℓ_1 that has distortion $O(1)$ for strings under our random model. Any such embedding must crucially use that the strings are random since Andoni and Krauthgamer [5] prove that embedding 1-non-repetitive strings into ℓ_1 requires distortion $\Omega(\log m / \log\log m)$. Andoni and Krauthgamer also study a related question, about approximately computing edit distance under a random model [6].

C Theoretical Guarantees of Our Algorithm

In this section, we estimate the number of strand comparisons performed throughout the execution of the algorithm. We analyze a serial version of the algorithm that slightly differs from the one described in Section 3. This version works as follows. It maintains a collection of clusters, which we will call groups to differentiate them from the clusters in the true clustering. The algorithm starts with singleton groups i.e., initially, every strand belongs to its own group. At every iteration t, the algorithm picks an anchor – a random string $a(t)$ of length L_A. Then, in each group g it picks a random strand $x_g(t)$, which we call the center of the group g. For every center x, the algorithm finds all substrings of length $L = L_A + L_H$ with prefix $a(t)$. These substrings are hash values for g. Denote the set of hash values by $H_g(t)$. The algorithm adds g to *all buckets* of a hash table indexed by $h \in H_g(t)$. (Note that the only difference between this algorithm and the algorithm we described earlier is that in that version we add g only to one of the buckets.) Then, the algorithm compares every two elements in each bucket and merges those groups whose centers are nearby with respect to the edit distance.

Theorem C.1. *There exists absolute constants $p_0 > 0$, $\beta_1, \beta_2, \beta_3 > 0$, such that for every $p \leqslant p_0$, $\varepsilon > 0$, integer $s \geqslant 1$, and sufficiently large n, m ($n > m^2$) the following holds. Let $L_A = \lceil \log_4 m \rceil$, $L = \min(\lfloor \frac{\log m}{6p} \rfloor, \log_4(n/m))$. Then, after $T = \frac{\beta_1 4^{L_A} \log(s/\varepsilon)}{m(1-2p)^L}$ steps, the algorithm recovers $(1-\varepsilon)$*

fraction of all clusters in expectation. The expected number of comparisons performed by the algorithm is upper bounded by

$$O\Big(\frac{n^2 m(1+\log(s/\varepsilon)/s)}{4^L(1-2p)^L}\Big).$$

If $L = \lfloor \frac{\log m}{6p} \rfloor$*, we get the bound*

$$O\left(\frac{n^2(1+\log(s/\varepsilon)/s)}{m^{\beta_2/p}}\right).$$

If $L = \log_4(n/m)$*, we get the bound*

$$O\Big(n^{1+2p} m^{2(1+p)}(1+\log(s/\varepsilon)/s)\Big).$$

Proof. As discussed before, the true clusters C_i are separated, so we never merge groups from different true clusters. Hence, every group g is a subset of some cluster C_i. Let us introduce some notation: Denote the set of groups a cluster C_i is split into at the beginning of iteration t by $\mathcal{G}_i(t)$; the set of centers chosen for these groups by $\text{center}_i(t)$, and the number of these groups/centers by $s_i(t)$. Note that $\mathcal{G}_i(0)$ is a set of singletons $\mathcal{G}_i(0) = \{\{u\} : u \in C_i\}$; $\text{center}_i(0) = C_i$; and $s_i(0) = |C_i| = s$. Let T be the total number of steps performed by the algorithm; and $\mathcal{G}_i(T)$, $\text{center}_i(T)$, and $s_i(T)$ be the set of groups, the set of centers, and the number of centers (respectively) in the cluster i at the end of the algorithm.

We need to show that the expected number of i such that $s_i(T) = 1$ is $(1-\varepsilon)n$ and then give a bound on the expected number of comparisons.

To analyze the algorithm we need to obtain lower and upper bounds on the probability that at one iteration two distinct centers u and v end up in the same bucket. For any two strands u and v, let S_{uv} be the set of all strings of length L that are substrings of both u and v; and let A_{uv} be the set of prefixes of strings in S_{uv} of length L_A. That is,

$$\begin{aligned} S_{uv} &= \{s \in \Sigma^L : s \text{ is a substring of } u\text{, and } s \text{ is a substring of } v\}; \\ A_{uv} &= \{\text{prefix of length } L_A \text{ of } s : s \in S_{uv}\}. \end{aligned}$$

Suppose that at iteration t, strands u and v are chosen as centers of two distinct groups g_1 and g_2. Then, the strands u and v are placed into the same bucket of the hash table if and only if the anchor $a(t)$ belongs to the set A_{uv}. This happens with probability $|A_{uv}|/4^{L_A}$, since the anchor $a(t)$ is a random string of length L_A, and there are exactly 4^{L_A} strings of length L_A. Recall that in this version of the algorithm, a strand may be placed not in one but a few different buckets if the anchor string occurs in it several times. The expected number of buckets that contain both u and v at step t is $|S_{uv}|/4^{L_A}$.

The sets A_{uv} and S_{uv} are random; and the quantities $|A_{uv}|$ and $|S_{uv}|$ are random variables. We estimate the expectation of $|S_{uv}|$ and $|A_{uv}|$.

Claim C.2. *Suppose* $4^{L_A} \geqslant 6m$*. Then the following bounds hold.*

1. *If* $u, v \in C_i$ *for some* i*, then* $\mathbb{E}|S_{uv}| \geqslant (m-L+1)(1-2p)^L$ *and* $\mathbb{E}|A_{uv}| \geqslant 1/6\ (m-L+1)(1-2p)^L$.
2. *If* $u \in C_i$*,* $v \in C_j$ *for* $i \neq j$*, then* $\mathbb{E}|S_{uv}| \leqslant m^2 4^{-L}$.

Proof. 1. If u and v belong to the same true cluster C_i, then they are noisy reads/copies of the same strand w. The strand w has length m and contains $(m-L+1)$ substrings of length L. Each of these substring is present in both u and v with probability $(1-p)^L \times (1-p)^L \geqslant (1-2p)^L$. Hence, the expected number of common substrings is at least $(1-2p)^L(m-L+1)$.

We now estimate $\mathbb{E}|A_{uv}$. The strand w contains $(m-L+1)$ prefixes of length L_A of substrings of length L. Let Z_a be the number of occurrences of a fixed "anchor" a (i.e., a substring of length L_A) in w that starts at least L characters before the end of the strand w. The expectation of Z_a equals $\mathbb{E}[Z_a] = (m-L+1)4^{-L_A}$, we denote this expectation by μ. A standard computation shows that the variance $\mathbf{Var}[Z_A]$ is upper bounded by $\mu + 2/3\mu$. Thus, $\mathbb{E}[Z_A^2] = \mathbf{Var}[Z_a] + (\mathbb{E}[Z_A])^2 \leqslant \mu + 2/3\mu + \mu^2$. Let $I(Z_a \geqslant 1)$ be the indicator of the event $\{Z_a \geqslant 1\}$. Then, by Cauchy–Schwarz,

$$\Pr(Z_a \geqslant 1) = \mathbb{E}[I(Z_a \geqslant 1)] = \mathbb{E}[I(Z_a \geqslant 1)^2] \geqslant 2\mathbb{E}[I(Z_a \geqslant 1)Z_a] - \mathbb{E}[Z_a^2] \geqslant$$
$$\geqslant 2\mu - (\mu + 2\mu/3 + \mu^2) = \mu/3 - \mu^2 \geqslant \mu/6.$$

Thus, the expected number of distinct anchors in strand w is at least $\mu/6 \times 4^{L_A} = (m - L + 1)/6$. Each of these anchors gets copied to u and v without errors together with the next $L_H = L - LA$ characters in w with probability at least $(1-2p)^L$. Hence, $\mathbb{E}|A_{uv}| \geqslant (m - L + 1)(1-2p)^{L_A}/6$.

2. Similarly, if $i \neq j$, then each u and v contains $m - L + 1 \leqslant m$ substrings of length L. The number of pairs of strings of length L – one from u and one from v – is at most m^2. The probability that two particular substrings are the same is 4^{-L} (as strings u and v are independent random strings in the alphabet Σ). Hence the expected number of common substrings of length L is at most $m^2 4^{-L}$. □

We would like to show now that the random variables $|S_{uv}|$ and $|A_{uv}|$ are concentrated around the mean w.h.p. That is true for $|A_{uv}|$ if $u, v \in C_i$ (see Lemma C.3). However, that is not true for $|S_{uv}|$ if $u \in C_i$, $v \in C_j$. To overcome this problem, we consider a sum of many random variables $|S_{uv}|$. We fix a cluster C_i and vertex $u \in C_i$ and bound the sum $\sum_{v \notin C_i} |S_{uv}|$ (see Lemma C.3).

Lemma C.3. *There exists an absolute constant β such that the following inequalities hold if $L < m/2$ and $L_A \geqslant \log_4 6m$.*

1. *If $u, v \in C_i$ for some i, then*
$$\Pr(|S_{uv}| \geqslant 1/10\, m(1-2p)^L) \geqslant 1 - \exp(-\beta m(1-4p)^L/L^2);$$
2. *If $u \in C_i$, $v \in C_j$ for some $i \neq j$, then*
$$\Pr\Big(\sum_{v \notin C_i} |S_{uv}| \leqslant \frac{2nm^2}{4^L}\Big) \geqslant 1 - m^2 \exp\Big(-\frac{\beta n}{4^L}\Big).$$

The proof of Lemma C.3 is similar to the proof of Claim C.2. We estimate the expectation as in Claim C.2 and then apply a concentration inequality. The only minor complication is that not all random variables in the sum we get are independent. Thus, in part (1), we use McDiarmid's Bounded Difference Inequality; and in part (2), we use the standard Chernoff bound, but we break the sum into m^2 sums of independent random variables. We omit the details here. We now proceed to the proof of Theorem C.1.

Correctness. We first show that the expected number of clusters completely recovered after T steps of the algorithm is $(1-\varepsilon)$. Consider a cluster C_i and $u, v \in C_i$. Suppose that u and v are centers of two different groups in C_i. If the algorithm places u and v in the same bucket of the hash table, which happens with probability $|A_{uv}|/4^{L_A}$, then the groups corresponding u and v are merged, since all clusters are separated (as discussed in Sections 3 and A). We say that C_i is good if for all $u, v \in C_i$ we have $|A_{uv}| \geqslant 1/10\, m(1-2p)^L$; otherwise, we say that C_i is bad.

Lemma C.3 asserts that the expected fraction of bad clusters is at most $\exp(-\beta m(1-4p)^L)$. Observe that

$$(1-4p)^L = \exp(-L\log(1/(1-4p))) \geqslant \exp(-5Lp) \geqslant \exp(-5/6 \log m) = m^{-5/6}.$$

Here we used that for a sufficiently small p, we have $\log(1/(1-4p) \leqslant 5p$. We also substituted $L = \lfloor \frac{\log m}{6p} \rfloor$. Hence, the expected fraction of bad clusters is upper bounded by $\exp(-\beta m^{1/6}/L^2)$, which is much less than ε for a sufficiently large m. Thus, we can ignore bad clusters.

Consider a good cluster C_i. At every iteration t, for any two centers $u, v \in \text{center}_i(t)$, the probability that the algorithm puts u and v into the same bucket is $|A_{uv}|4^{-L_A}$, which at least

$$\alpha = \frac{m(1-2p)^L}{10 \cdot 4^{L_A}}.$$

Therefore, using Lemma C.5 from Section C.1, we obtain the following bound on the expected number of non recovered clusters:

$$s(1-\alpha)^T = s\Big(1-\frac{m(1-2p)^L}{10\cdot 4^{L_A}}\Big)^T \leqslant \varepsilon/2.$$

The last inequality holds, since

$$T = \frac{\beta_1 4^{L_A}\log(s/\varepsilon)}{m(1-2p)^L}.$$

Number of Comparisons. We now upper bound the total number of comparisons. We divide all strand comparisons into two types: internal and external. We say that a comparison of u and v is internal if u and u belong to the same true cluster C_i; we say that it is external, if u and v belong to two distinct clusters C_i and C_j. It is easy to see that the total number of internal comparisons for a cluster C_i is bounded by $|C_i|-1=s-1$, since after $s-1$ comparisons the algorithm will merge all vertices in C_i into one group. Thus, the total number of internal comparisons is $k(s-1)<n$. Below, we prove Lemma C.4 that gives a bound on the number of external comparisons. This bound together with the bound on the number of internal comparisons gives us the desired result. $\square$

Lemma C.4. *The expected number of external comparison performed by the algorithm in T steps is upper bounded by*

$$O\Big(\frac{n^2m(1+\log(s/\varepsilon)/s)}{4^L(1-2p)^L}\Big).$$

Proof. Fix a cluster C_i. We estimate the expected number of external comparisons between strands in C_i and outside of C_i. Consider a step t of the algorithm. If u is chosen as a center of a group in C_i, then the expected number of external comparisons between u and strands v outside of C_i is upper bounded by $4^{-L_A}\sum_{v\notin C_i}|S_{uv}|$ (note: as we discussed earlier $4^{-L_A}|S_{uv}|$ is the expected number of buckets in which both u and v are placed at iteration t if u and v are centers). Here, we use a conservative estimate: instead of counting only strands v that are centers of groups outside of C_i we count all v's outside of C_i. Summing over all centers $u\in \text{center}_i(t)$, we get a bound on the expected number of comparisons between centers in C_i and other centers:

$$\begin{aligned} 4^{-L_A}\mathbb{E}\Big[\sum_{u\in \text{center}_i(t)}\sum_{v\notin C_i}|S_{uv}|\Big] &\leqslant 4^{-L_A}\mathbb{E}\Big[|\text{center}_i(t)|\max_{u\in C_i}\sum_{v\notin C_i}|S_{uv}|\Big] & (1)\\ &= 4^{-L_A}\mathbb{E}\Big[s_i(t)\sum_{v\notin C_i}|S_{uv}|\Big]. & (2)\end{aligned}$$

Let $\mathcal{E}_u$ be the event $\{\sum_{v\notin C_i}|S_{uv}|\leqslant 2nm^2/4^L\}$, and

$$\mathcal{E}_{C_i} = \cap_{u\in C_i}\mathcal{E}_u = \Big\{\max_{u\in C_i}\sum_{v\notin C_i}|S_{uv}| \leqslant \frac{2nm^2}{4^L}\Big\}.$$

Let I_i be the indicator of the event $\mathcal{E}_{C_i}$. By Lemma C.3, $\Pr(\neg\mathcal{E}_u)\leqslant m^2\exp(-\beta n/4^L)$. Using the union bound, we get $\mathbb{E}[1-I_i]=\Pr(\neg\mathcal{E}_{C_i})\leqslant sm^2\exp(-\beta n/4^L)\leqslant 4^{-L}/s$. The last inequality follows from the bound $L\leqslant \log_4(n/m)$ for sufficiently large m. We now consider two cases: $I_i=1$, then we bound the expected number of comparisons using (2); and $I_i=0$, then we bound the number of comparisons by sn (this is the maximum possible number of comparisons). We get the following bound on the number of comparisons between centers in C_i and all other centers at iteration t:

$$\begin{aligned} \frac{1}{4^{L_A}}\mathbb{E}\Big[I_i\cdot s_i(t)\cdot\sum_{v\notin C_i}|S_{uv}|\Big]+\mathbb{E}\Big[(1-I_i)\cdot sn\Big] &\leqslant \frac{1}{4^{L_A}}\mathbb{E}\Big[I_i\cdot s_i(t)\cdot\frac{2nm^2}{4^L}\Big]+\mathbb{E}\Big[(1-I_i)\cdot sn\Big]\\ &\leqslant \frac{2nm^2}{4^{L+L_A}}\mathbb{E}[s_i(t)]+\frac{n}{4^L}.\end{aligned}$$

We now sum up this bound over all steps $t=0,\dots,T-1$ of the algorithm. By Lemma C.5, $\mathbb{E}\big[\sum_{t=0}^{T-1}|s_i(t)|\big]\leqslant T+s/\alpha$, where $\alpha=m(1-2p)^L/(10\cdot 4^{L_A})$. Hence, the expected number of external comparisons for a cluster C_i is upper bounded by

$$\frac{2nm^2}{4^{L+L_A}}\cdot\big(T+\frac{10s\cdot 4^{L_A}}{m(1-2p)^L}\big)+\frac{nT}{4^L} = O\Big(\frac{Tnm^2}{4^{L+L_A}}+\frac{snm}{4^L(1-2p)^L}\Big).$$

Summing up this bound over all k clusters C_i, we get the bound:

$$O\Big(\frac{Tn^2m^2}{s4^{L+L_A}} + \frac{n^2m}{4^L(1-2p)^L}\Big) = O\Big(\frac{n^2m(1+\log(s/\varepsilon)/s)}{4^L(1-2p)^L}\Big).$$

□

C.1 Single Cluster Dynamics

In this section, we analyze the evolution of a single cluster C_i from the underlying true clustering throughout the execution of the algorithm. We obtain the desired bounds, we will only rely on the following fact: The probability that any two groups in C_i are merged at step t is at least α. Note, however, that the events $\{g_1$ and g_2 are merged at step $t\}$ and $\{g_2$ and g_3 are merged at step $t\}$ are not independent.

Lemma C.5. *The following bounds hold.*

1.
$$\Pr(S_i(T) > 1) \leqslant (s-1)(1-\alpha)^T.$$

2.
$$\mathbb{E}\Big[\sum_{t=0}^{T-1} s_i(t)\Big] \leqslant T + \frac{s-1}{\alpha}.$$

Proof. 1. Pick a designated vertex u in C_i. For every v the probability that u and v are in two distinct groups after t steps of the algorithm is at most $(1-\alpha)^t$ since at every step the probability that the group containing u and group containing v are merged is at least α. Thus, the expected number of vertices that are not in the group containing u is at most $(s-1)(1-\alpha)^t$. This is also a bound on the number of groups not containing u. Hence, $\mathbb{E}[s_i(t)] \leqslant 1 + (s-1)(1-\alpha)^t$, and, consequently, $\Pr(s_i(t) > 1) \leqslant \mathbb{E}[s_i(t) - 1] \leqslant (s-1)(1-\alpha)^t$.

2. Using the bound $\mathbb{E}[s_i(t)] \leqslant 1+(1-\alpha)^t$, we get $\sum_{t=0}^{T-1}(1+(s-1)(1-\alpha)^t) \leqslant T+(s-1)/\alpha$. □

D Outlier Analysis

For this proof, we analyze a slightly simplified version of our algorithm that does not use signatures for merging. A similar result holds for the original algorithm, but it is more complicated to state (since it depends on $\mathcal{D}_{s,p,m}$), and this complexity does not add new insight. In particular, for this section, we will set $\theta_{low} = -1$. The other parameters of the algorithm can be arbitrary, except for r which depends on the separation (and is mentioned in the theorem statement).

We need some definitions for the proof.

Definition D.1. Let $\mathbf{C} = \{C_1, \dots, C_k\}$ be $(r, 2r)$-separated. A point z *touches* a cluster C_i if $d_E(z, x) \leqslant r$ for any $x \in C_i$. For a set O of outliers, say a cluster C_i is *untouched by* O if no point in O touches it.

Proof of Theorem 4.2. We show that at most $\varepsilon' k$ terms in the sum in $\mathcal{A}_\gamma$ differ for $\widetilde{\mathbf{C}}$ versus $\widetilde{\mathbf{C}}'$. This follows from two simple claims:

1. If C_i is untouched by O, then the ith term is the same in the sums for both $\mathcal{A}_\gamma(\mathbf{C}, \widetilde{\mathbf{C}})$ and $\mathcal{A}_\gamma(\mathbf{C}, \widetilde{\mathbf{C}}')$.
2. Each outlier in O can touch at most one cluster in $\mathbf{C}$.

To see the first of the these claims, notice that if a cluster C_i is untouched, then $d_E(x, z) > r$ for any $z \in \mathsf{O} \cup (S \setminus C_i)$. Therefore, no element of C_i will ever merge with an element outside of C_i, and the output of the algorithm will be the same with respect to C_i on both inputs S and $S \cup \mathsf{O}$. The second claim follows directly from the $(r, 2r)$-separated property. Indeed, if an outlier z has $d_E(x, z) \leqslant r$ for any point $x \in C_i$, then it must be that $d_E(y, z) > r$ for all $y \in C_j$ with $j \neq i$. Putting these claims together, at most $\varepsilon' k$ terms differ in the sums for $\mathcal{A}_\gamma$ and thus the accuracies differ by at most ε', since each term is at most 1. □

E Additional Information about Datasets and Experimental Setup

E.1 Computing Environment and Implementation Details

We use dedicated virtual machines on Microsoft Azure within a single region and virtual network. They machines have Azure size H16mr. The specifications are as follows: Intel E5-2667 V3 3.2 GHz processors, utilizing 224GB DDR4 memory and 2TB SSD-based local storage. They have a dedicated RDMA backend network enabled by FDR InfiniBand network. The Linux operating system uses image Centos 7.1-HPC.

We implemented out algorithms using C++. We used the MPICH MPI-3 compilers and the Intel MPI 5.2 runtime library. We compiled with the -O2 flag. For MPI, we allocated a single rank per core, and the large communication steps of our algorithm are implemented with non-blocking windows, which support RDMA.

E.2 Real Datasets Used for Evaluation

We provide details about the real datasets that we used for experimental evaluation. The real data came from a DNA data storage system presented by Organick et. al. [45]. We explain the way we have processed their sequencing data to generate our datasets. In particular, we explain the methods we used to produce a gold standard on which we evaluate our algorithm and Starcode.

Overall, the data from Organick et. al. [45] is the output of an Illumina NextSeq machine. They prepared and sequenced molecules of synthetic DNA that store encoded data. The details of wetlab preparation and amplification can be found in the paper by Organick et. al. [45].

To generate a gold standard for clustering, we must find the true mapping between references and reads. To do this, we used a standard biological alignment tool, the Burrows-Wheeler Aligner (BWA) [39]. We run BWA with 10 threads using the command

```
bwa mem -t 10 [reference file] [read file]
```

BWA first indexes the references (the expected synthesized DNA) to create a database of references. After the references are indexed, BWA compares the references to all reads in order to find the best mapping between the references and reads. Each reference maps to many reads, which will be the basis for the clusters.

BWA also outputs an alignment between reads and references in the “bam” file format. From this, we extract the reads that aligned successfully. More precisely, each line in a bam file contains several fields, one of which is the read. Another field in the bam file keeps track of the most similar reference for this read, or leaves this field empty if no references are found to align. For a read that aligns, the file also identifies the best alignment (that minimizes edit distance between the read and reference). We discard lines that do not align to any reference. Since reads of DNA often contain errors, as well as additional nucleotides before and after the expected strand, we used the alignment results to extract the portions of the reads that align to the references. Finally, from the BWA output, we know which reference each read is most similar to. Thus, we use this to create gold standard clusters.

We note that in our experiments, the clustering algorithms do not have access to the references, so the clusters generated using BWA should have much higher quality than anything produced by a clustering algorithm that does not have access to the references. Therefore, the alignments serve as a good gold standard.

The datasets were generated from a single sequencing run, which in total produced 58M reads after processing. The set of references for this run corresponds to about 40MB of data stored in DNA. Additionally, using the original file information, we are able to separate out three smaller datasets, each corresponding to a single encoded file. These represent a movie file, a text file, and a music file, respectively. The sizes of the these smaller files appear in Table 1.

We remark that in practice, any true DNA storage system has an additional challenge. The data retrieval pipeline must identify the noisy reads from the sequencer output. The complication arises because the sequencer may append relatively long strings of random characters on either end of each read that it outputs. Since systems cannot align to true references, they must use additional information, such as a known substring, to extract a high-quality subsection of the read for clustering.

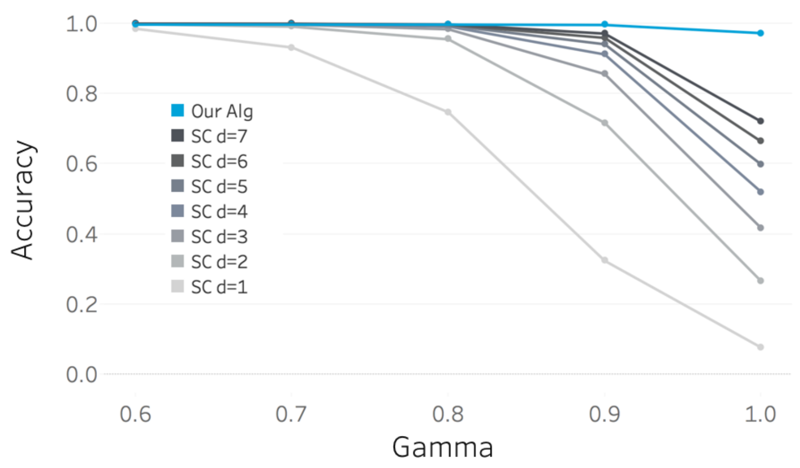

Figure 4: For the 12M real reads dataset, the graph plots the accuracy $\mathcal{A}_\gamma$, for varying γ, of our algorithm versus Starcode (with different distance threshold settings for Starcode). Higher is better. Tests were run on one processor, 16 threads.

E.2.1 Synthetic Datasets Used for Evaluation

We follow Definition 2.3. The datasets with generated using Python 3.6, using the default random library. We implemented the uniform noise generation with "random.uniform", picking insertion, deletion, or substitution, each with equal probability for 4% total chance of error. We picked insertion and substitution characters uniformly at random in $\{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}\}$.

E.2.2 Sources of Errors

In DNA data storage systems, errors arise not only from sequencing, but also from synthesis and amplification. This leads to a higher overall error rate than sequencing alone, with less than a ten-fold difference between substitution and insertion/deletion rates. In particular, the majority of reads contain at least one insertion or deletion, and therefore the clustering algorithm must be tailored to edit distance. Finally, although Illumina error rates may be low, future technologies such as Nanopore sequencing will require clustering algorithms that are robust to much higher error rates.

F Accuracy Justification

We expound on our definition of accuracy and its relationship to DNA storage. Recall that the goal of clustering is to find the groups of reads that came from the same reference. Then, using these clusters, a method called trace reconstruction is used to determine the most likely reference for each group of noisy reads. The effectiveness of trace reconstruction depends on the number of *traces*. Therefore, larger clusters are clearly better. When the size of the original clusters varies, it is better to have more traces from small clusters, therefore we use this parameter γ to measure the fraction of recovered strings in a cluster.

Trace reconstruction methods do not have a built-in way to handle false positives. Fortunately, it is easy to check false positives because we know the threshold r. In particular, we assume that any clustering methods will check false positives before outputting the clusters.

G Additional Experimental Result

We include an accuracy comparison in Figure 4 for the 12M real reads dataset. This corresponds to the times reported for this dataset in Figure 2a in Section 5. We show the output of Starcode for distance parameters 1 through 7 (the setting of distance 8 did not finish in 28 hours).