Index-based clustering

Research Report

Yoav Erez
Ohad Einav

June 2022

Table of Contents

Overview

Existing information storage technologies are having difficulty keeping up with the mass amounts of data in today's world.

For this purpose, the field of DNA based storage arose, providing a method to store bits of information in synthesized DNA, and thus reducing the size of the information storage while increasing its effectiveness.

Such DNA synthesis methods have emerged as promising options, with theoretical information density of multiple orders of magnitude more than magnetic tapes.

However, this approach is not without its setbacks. DNA synthesis and maintenance is an error-prone process, with information strands getting modified at high error rates. This can result in loss of critical information, if not handled correctly.

One solution to this problem offers to multiply each strand of information multiple times, and then cluster those multiples when extracting all of the DNA strands. To make things simpler, an identifying index is prepended to each multiplied strand.

This approach lends an opening to the family of clustering algorithms, where accuracy and time efficiency are both key.

In our research, we expound on previous work done in DNA clustering, and show significant improvements on the current benchmarks, both in accuracy and in time efficiency.

By focusing on the singleton clusters, we were able to significantly reduce the outliers, resulting in leaps in performance with only a marginal runtime increase.

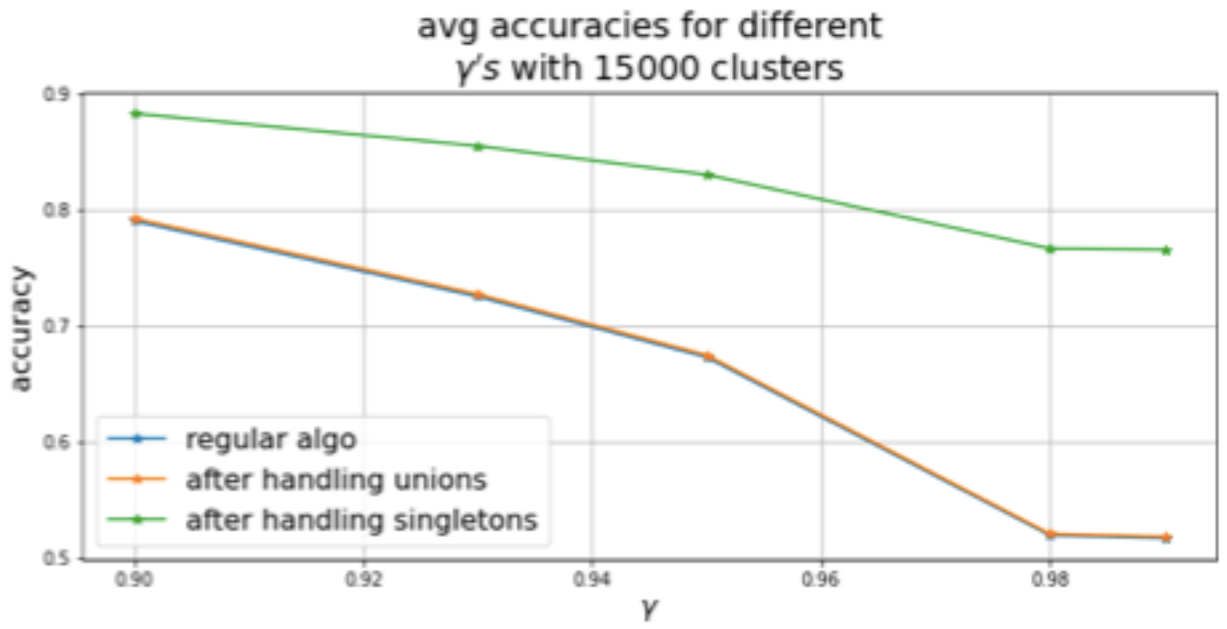


Figure 1: Performance of the different algorithms on a dataset of 15000 clusters, with an avg of 40 reads per cluster and strand lengths of 150. The gamma parameter is an evaluation metric that affects how accurate a clustering must be to be considered an effective clustering.

Preliminary analysis showed the singleton clusters as significant outliers that affected the overall algorithm performance. Many of these are reads that were not able to be successfully merged into their original clusters because of high error rates pushing them out.
Performing extra iterations to deal with these singletons proved to be highly effective.
As is shown in Figure 1, the incremental gains after handling singletons on a sample set of 15K clusters are near +20%, while accounting for a runtime increase of only 25%.

This report details the research done to examine possible lifts in the performance of current DNA clustering algorithms, when incorporating index to the DNA strands metadata.
We will outline the methodologies of our research and display our full results on the gains we have reached in DNA clustering.

Project motivation and goals

Evaluation

Two of the main operating points by which our clustering solution are evaluated are:
1) An accuracy metric presented by *Raschtian et al.* in their paper: *Clustering billions of reads for DNA data storage*[1]. This metric looks at each cluster provided by the solution and calculates the fraction of reads in the cluster that are indeed copies of the same DNA strand from the original data.
If this fraction is higher than a hyper-parameter γ, then we consider that cluster to be accurate.

2) The runtime complexity of the algorithm, including its scalability to many strands and large cluster sizes.

The goal of this project was to determine if we can leverage index information, as well as the core data of the reads, to provide a significant lift in the bottom-line accuracy and efficiency of current DNA clustering algorithms.

[1] https://papers.nips.cc/paper/6928-clustering-billions-of-reads-for-dna-data-storage

Background: DNA Data storage:

Motivation

In a world flooded with data, figuring out where and how to store it efficiently becomes a larger problem every day. The prevailing long-term storage method is to write data to heavy and sizeable reels of magnetic tape.
A seemingly exotic solution to this problem might turn out to be one of the best, archiving information in DNA molecules.
While the expense of DNA storage is still a work in progress, it is more energy efficient and longer lasting. In the proper environments, DNA remains stable for decades and doesn’t require maintenance.

Perhaps the greatest attraction of DNA storage, however, is simply the sheer amounts of data that can be stored in small spaces.
With an information density of 10^9 GB/ per cubic millimeter,
DNA can archive a staggering amount of information in an incredibly small volume.

Synthesizing data into DNA

Properly encoding and extracting information through DNA requires several steps. The 1st is to convert the file being stored into sequences of four letter-combinations of A, C, G, T, which is simply a binary to base-4 conversion.
Next, the file must be split up into many original DNA strands or references. These references are then copied multiple times to provide insurance against errors in the synthesis.
Each one of these copies is a read of a particular reference.
These reads are then stored in in individual DNA strands in some container.

Sequencing

To retrieve the data, the DNA is accessed using next-generation sequencing, which results in several noisy reads of each originally synthesized reference.
Sequencing is the process of determining the order of nucleotides in the DNA, which makes up the base-4 information that is being stored.

Clustering

Once all the reads of all the references are extracted through sequencing, the goal is to recover the unknown references from the observed reads.
The first step, which is the focus of this research, is to cluster the reads into groups each of which should be the set of noisy copies of a particular reference.

The output of the clustering is then fed into some consensus finding algorithm, which predicts the most likely reference to have produced each cluster of reads.

Current Solutions: Hash-based Clustering

Clustering many reads can be a very runtime-inefficient and unscalable process. Since the size of the clusters are fixed, the number of clusters grows linearly with the input size. Therefore, any algorithm requiring $\Omega(n^2)$ time would be too slow for large-scale datasets.
Additionally, algorithms must be robust to outliers, ruling out many other clustering methods.

Microsoft Research paper

In 2017, in the paper *Clustering billions of reads for DNA data storage*, *Raschian et al*. present an efficient hash-based method to cluster large-scale data.
They outline edit distance as the metric by which to cluster the reads together.

Edit distance between 2 reads represents the number of additions, deletions, or substitutions of letters that are required to make the reads identical.
This accurately portrays difference in the sequencing error-rate of 2 reads of the same reference.

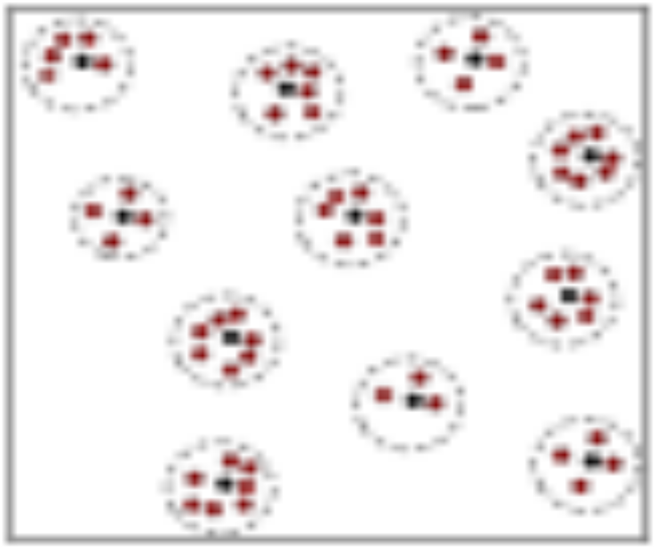


Figure 2: from the paper, DNA storage datasets have many small clusters that are well separated in terms of edit distance.

Hashing for edit distance

A critical property of Edit distance is that when two reads $x, y, \in \Sigma^m$ have an edit distance at most r, then they must share a substring of at length at least $\frac{m}{r+1}$. However, these matching substrings may appear in different locations.
An approximate solution presented by the paper is to design a hash family based on finding these matching substrings quickly.
Informally, for parameters $w, l,$ the hash picks a random "anchor" a of length w, and the hash value for x is the substring of length $w + l$ starting at the first occurrence of a in x.

Iteratively merging clusters based on representatives

With the aforementioned hashing algorithm in mind, the reads are then iteratively merged into clusters as follows:

The clusters are initialized as singletons according to the number of reads.
During each step, a representative is sampled at random from each cluster.
These representatives are then bucketed by a hash function chosen uniformly from the hash family.
Then, for each pair of representatives in the bucket, based on 3 parameters, $r, \theta_{low}, \theta_{high}$, the algorithm will merge the clusters of these representatives if one of the following conditions hold:

- The hamming distance between the binary signatures of the reads is lower than θ_{low}.
- The hamming distance between the binary signatures of the reads is lower than θ_{high}, and additionally, the edit distance of the two representatives is less than r.

These conditions are checked in order, so as not to unnecessarily perform the more expensive calculation of edit distance between the reads.

After a predetermined amount of merging steps, the algorithm outputs the final merged clusters.

Performance

The researchers measure the accuracy of the algorithm's output with relation to the original "true" clustering The Accuracy measures the number of clusters in the algorithm output that overlap with some cluster in the original clustering in at least a γ-fraction of elements while containing no false positives[2] .

[In the case of DNA storage, these clusters have a natural representation as each cluster should represent all the noisy copies of a single reference. This interpretation justifies the need for high accuracy. Therefore, the accuracy threshold should be very high.
Indeed, throughout our research the γ-values were set between 0.9-0.99]

[2] In our project, we tweaked this metric to be more lenient and allowed false positives, so long as the clusters contained a high fraction of shared elements ($> \gamma$)

Their algorithm claims to outperform the previous SOTA clustering methods while achieving a 1000x speedup. They claim to cluster 5B reads with 99% accuracy in 46 minutes on 24 processors.

Of course, these results are meaningless without mentioning the error rates involved in the simulations of the noisy copies in the dataset, as well as the algorithm's robustness to various cluster lengths and strand sizes.

In our simulations, we doubled the error rates and worked on clusters with average sizes of 40, more than double the sizes of the datasets dealt with in these performance claims.

We ran the paper's algorithm with our additional methods, and then compared the results side by side on the same dataset.

Research Methodology – Theory and Approach

Approach overview

Preliminary analysis on the current solutions to clustering showed three key points:

1) When evaluating on a dataset with a considerably higher error rate, the performance of the algorithm deteriorates significantly.
2) The algorithm's performance is highly sensitive to the parameters and conditions set forth when looking to compare the representatives of two different clusters
3) The accuracy of the cluster output is inversely proportional to the number of singleton clusters that remain at the end.

For these reasons, we took an aggregative approach: Run the current clustering algorithm on the reads, and then continue to fill in the gaps at the pain points where the original algorithm did not perform well.
We performed additional iterations on the Union (merging) of clusters as well as special treatment to the singletons that remained at the end of the original algorithm's runtime.

Additionally, special conditions tailored to our dataset were set for comparing the representatives of clusters[3]

The Dataset

For the dataset upon which to evaluate the results, we used the DNA storage simulator[4], evaluating inputs of 3K, 6K, 9K, 15K, and 50K strands.

- The error rates of these simulations were 2x the error rate of the *MinIon* simulation
- The strand lengths were constant at 150
- The cluster sizes were an average of 40 strands and max size 50
- Index metadata was prepended to the strands, such that the copies of each reference have identical indices

Handling Unions

When evaluating the original algorithm's merging behaviors, we noticed that there may be some invalid merges, i.e., merges between clusters that represent different reads.

To handle these "unwanted unions", we manually broke apart clusters that were suspect.

As a 1st step, we focused in on the clusters that were above average in size.

For each one of those clusters, the following process was executed:

- Find the 2 reads in the cluster that are farthest in Hamming Distance, and initialize them as 2 "group leaders"
- Separate each remaining read from the cluster into one of the groups based on the distance from the leaders.
  Whichever leader the read is closest to will put them together.
- For each group, evaluate the average hamming distance of the reads with themselves.
- If the average distances differ by more than some constant $thresh$, then we split the cluster into 2

[3] Conditions when comparing the representatives of clusters and singletons:

- Hamming distance of reads must be < 2/3 of the read size
- Edit distance of the indices must be < 1/3 the index size

[4] https://github.com/gadihh/DNASimulator/wiki

Handling Singletons
After iteratively merging the clusters using the original algorithm, there were still many singleton clusters remaining:

Number of strands	Remaining singletons
3,000	500
6,000	1,000
9,000	1,800
15,000	5,200
50,000	28,000

These singleton clusters were bringing down the overall accuracy greatly, so special attention was required to merge them to their respective clusters.
The extra attention, however, needed to be time-efficient to be considered as having an overall positive impact on the algorithm.

Unwanted Singletons
We defined singleton clusters as "unwanted singletons" if the size of the original cluster that the singleton read belongs to is greater than 2.
This definition helps us to more accurately evaluate our singleton reduction algorithm and have it better correlate to the overall clustering accuracy.

Method 1: Iterating over all cluster for each singleton

The first method we tried, was to manually iterate over each singleton and compare it to the representative of each cluster according to the conditions for comparing clusters that were set in the original algorithm. This provided significant lifts in accuracy, while absorbing significant damage in run-time. Seeing as the runtime complexity of this solution is $\Omega(\frac{input^2}{cluster\_size})$, this wasn't very scalable.

Results

AVG TIME IN MINUTES	**3000 strands**	**6000 strands**	**9000 strands**
Regular algorithm	0.7565	1.8915	2.9578
With union handling	0.056	0.112	0.1705
With singleton handling	5.7365	25.917	62.139

AVG ACCURACY	**3000 strands**	**6000 strands**	**9000 strands**
Regular algorithm	76%	75%	73.5%
With union handling	76%	76%	74.5%
With singleton handling	88%	87%	86.5%

Method 2: Hashing singletons and clusters

While Method 1 experienced pleasant gains in the accuracy lift after handling singletons, the additional iterations were not nearly efficient enough for our purposes.

For this reason, we adopted a hash-based approach, along the lines of what proved effective for the original clustering solution presented by Microsoft research.

For all trials, the general method was the same, differentiating only in the type of hashing that we used.
Hash each singleton into our hash map, any number of times.
Iterate over the representatives of each cluster, and calculate their hash value as well. For each singleton keep a list of all the representatives (i.e., clusters) that are candidates for this singleton. From the list of candidates, we then find the cluster most suitable to merge this singleton into, based on the original cluster comparing conditions.

Trial A – hashing by a random subset of data

The first trial was to create a hash function for each singleton based on a random subset of indices. We choose 5-10 indices in the range of the strand length, and the values at those indices are the hash key for that particular read.

Trial B – hashing by a random subset of binary signature

The second trial was to create a hash function for each singleton based on a random subset of indices to create subset of his binary signature. We choose 5-10 indices in the range of the binary signature length, and the values at those indices are the hash key for that particular read.
In order to allow little error space, we made t different sub signatures for each singletons. Therefore, each cluster representatives had t opportunities to become a candidate for a singleton.

Trial C – hashing by a random subset of the data with padded hash keys

We then tried to hash multiple keys for each singleton. Choosing a starting index at random we choose the sequence of 10-20 letters from that starting index in the read.
We then hash all the values that have a hamming distance within 2 of this candidate sequence.

If l denotes the length of the hashed sequence, and k the maximum condition for hamming distance, then the number of keys to be hashed for the read is $\binom{l}{k}$

Trial D – hashing using Microsoft research’s hashing algorithm

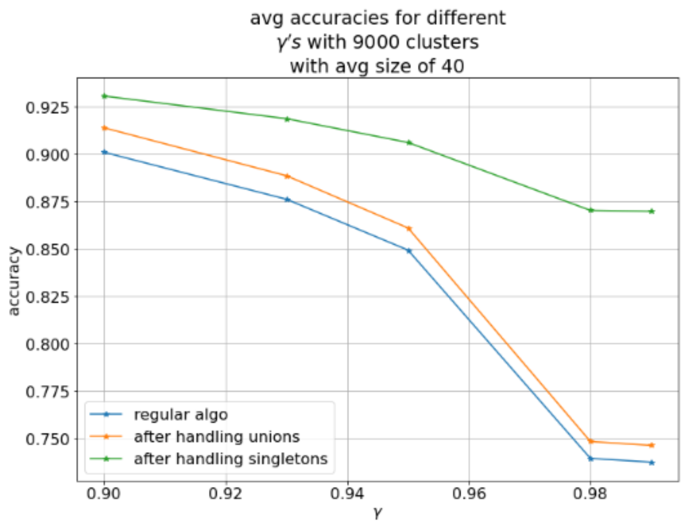
We then decided to use Microsoft hash function to hash the singletons. As done in the previous trials to get an error space, we initialized many variations of hash family (explained in “Hashing for Edit Distance” above), and used them to find valid candidates from the representatives of the clusters.
The hyperparameters of the hash functions to choose were:

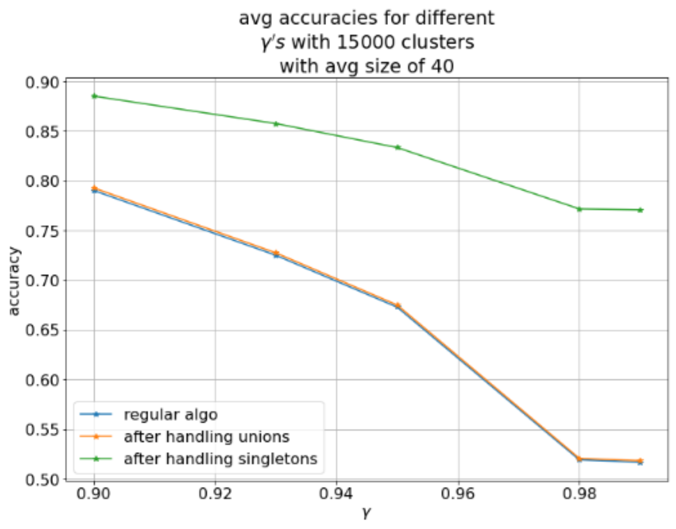
1. Number of hashes to make on random substrings of the singleton reads.
2. Length of the hash

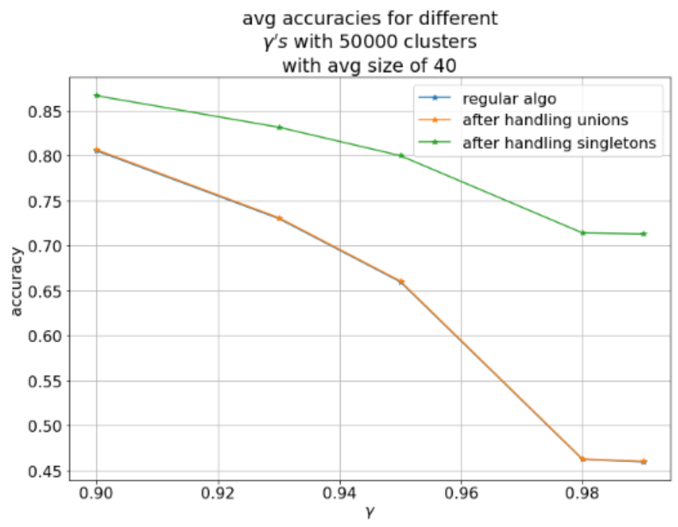
Results

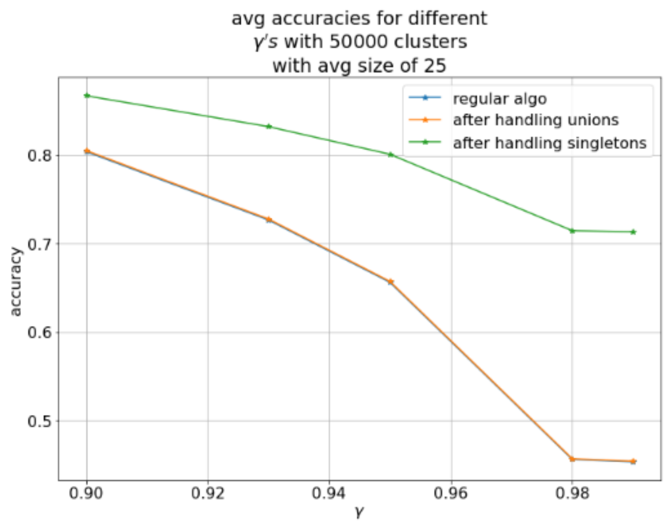
In order to measure the "goodness" of our trials we had to check two properties:
1. How many of the unwanted singletons (see Unwanted singletons section above) had the real cluster in their candidates list.
2. The average sizes of the candidates. If this number is large, then we probably won't improve the time performance that needed for efficient clustering.

All the trials we have tried show good results in the second property. However, only trial D shows good results for the first property.
Trial D produced good results in quick time after one epoch. Therefore, we decided to see what happens if we run this approach for multiple epochs until the rate of singletons reduction stagnates.
The following plots shows the accuracies for various numbers of clusters:



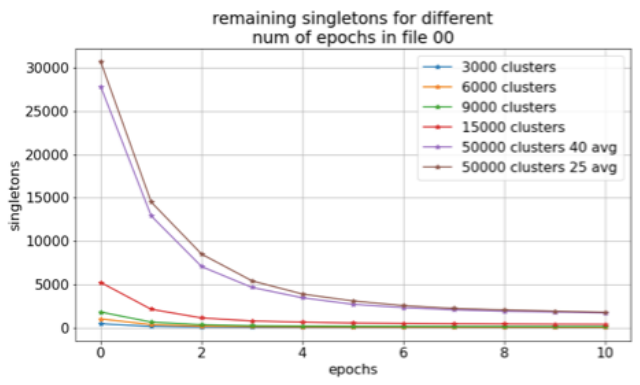


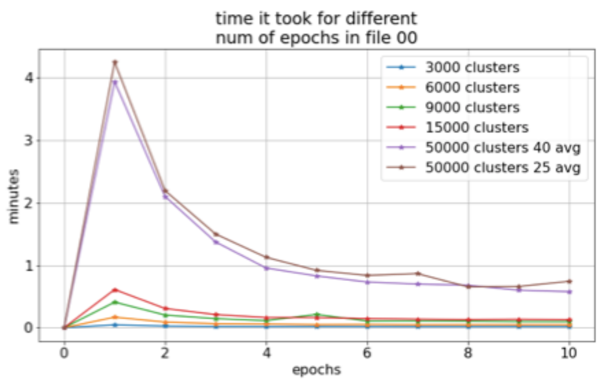




In the experiments on the largest input sizes (50K clusters), after only 2 epochs, we see an accuracy improvement of ~25% at gamma rates above 0.98.

Let's look at the time cost of these improvements:





Comparing our new runtime and performance to the original algorithm:

AVG TIME IN MINUTES	**9000 strands**	**15,000 strands**	**50,000 strands**
Regular algorithm	2.62	5.407	24.656
With union handling	0.132	0.260	1.085
With singleton handling	0.478	0.838	6.187

AVG ACCURACY on $\gamma = 0.99$	**9,000 strands**	**15,000 strands**	**50,000 strands**
Regular algorithm	74%	52.5%	46%
With union handling	75%	52.5%	46%
With singleton handling	87%	77.5%	71.5%

There are a few things to notice before diving into the bottom-line results:

1. By comparing the experiments on the 9,000 strands input, we observe that we managed to reduce the runtime from method 1 by 155x, or 2 orders of magnitude.
2. This efficiency does not come at any cost of accuracy, as our algorithm's performance remained stable.

3. The algorithm is scalable – When evaluating on 50K clusters and variating avg. cluster sizes (40,25), the runtime for the accuracy increase due to singleton handling does not exceed 25% of the original algorithm's runtime.

This leads us to our bottom-line:

From our experiments on inputs greater than 15,000 strands, at gamma rates above 0.98, on an error rate of 2x the industry standard, we produced a 25% increase on the overall accuracy, while maintaining a runtime increase upper bound of only 25%.

Furthermore, this production seems to be scalable to large datasets and higher error rates.

Future work

Our improvements, while significant, did not manage to properly leverage the index information imbued in the read metadata.
Indices were not spaced well enough to overcome the extreme error rates.

We believe that future research can try alternative indexing approaches to further improve our clustering methods.

Acknowledgements

Thank you to Avital Boruchovsky and Dvir Ben Shabbat for the close guidance, weekly meetings, and assistance throughout this research.

Links

- Our Github link
- Clustering billions of reads for dna storage - paper