CSE 547 SP20: Clustering Covid-19 Viral Genomes and Spike Protein Sequences

Aishwarya Mandyam, Cheng Ni, Jack Khuu
Paul G. Allen School of Computer Science and Engineering
University of Washington
Seattle, WA 98195, USA
[aishrm2, chengni, jackkhuu] @cs.washington.edu

Abstract

COVID-19 brought researchers from around the world to study its viral genomes in order to better understand the virus. GISAID has a dataset that contains COVID-19 viral genomes and spike protein sequences. However, many of these sequences are not high quality and the dataset in general presents significant computational challenges. We use clustering to answer broad domain questions related to the spread of the virus within a city and between regions of the world. We find that custom clustering algorithms are likely to group sequences together based on the timestamp of the sample and the location of the sample, and that these results correspond to and agree with news and travel patterns for the spread of coronavirus.

1 Introduction

By the end of 2019, the world was interrupted by the beginnings of a global pandemic. Months later, researchers are still trying to understand how COVID-19 evolves and spreads using the limited dataset available. The most prominent dataset publicized contains data related to mortality and survival rates [1]. However, there is a lesser known dataset by GISAID [2], an organization that usually collects data on Influenza, which has viral genome samples from Covid-19 patients from around the world. This project aims to understand existing relationships in this dataset using clustering algorithms.

2 Prior Work

There is extensive work studying the effectiveness and use of clustering algorithms, but there are less algorithms that are tailored to clustering biological sequences. We look at four existing algorithms that cluster DNA sequences of varying lengths. Some optimize for computational efficiency, while others look at highly customized distance functions. A brief description of each algorithm follows.

2.1 Starcode

Zorita et al. proposed an algorithm to cluster sequences within a particular Levenshtein distance threshold using pairwise searches for all pairs in a dataset [3]. The model uses a trie structure to store the sequence prefixes to reduce the search space and significantly speed up the process. This reduces the computation of edit distance and makes clustering more efficient.

2.2 MeSHClust

James et al presents a tool for clustering DNA sequences with an approach adapted from the mean shift algorithm [4]. It is guaranteed to converge within finite steps and is able to produce global alignment

Preprint. Under review.

scores using alignment-free methods. The model consists of a combination of a classification step and a clustering step so that the users don't need to specify the number of clusters to generate.

2.3 Clustering Billions of Reads

Rashtchian et al. focuses on high cardinality and dimensional clustering of synthetic DNA for data storage [5]. They use clustering for error-correction in the data retrieval step of DNA storage. The data is a synthetically generated noisy representation of each sequence resulting in billions of clusters that linearly scale with the size of the data size. They created an agglomerative algorithm that uses edit distance as the cost function and can handle clustering in a distributed fashion. The key idea is that data is ingested into the model in parallel and items are hashed into buckets through local computation and redistributed through global communication for clustering.

2.4 Fuzzy Integral Similarity

Saw et al. discusses an alignment free method of sequence clustering to explore faster sequences analysis algorithms when dealing with large datasets and computation time is a key constraint [6]. It makes use of the fuzzy integral with Markov chain and computes frequencies of occurrence of all possible nucleotide pairs from each DNA sequence. The model generates a symmetric distance matrix that can be further used to construct a phylogenetic tree using k-mer based similarity score.

2.5 Contribution

This project applies existing methods to the GISAID dataset to uncover existing relationship. The project also customizes these existing methods to ensure that they are better suited to the GISAID dataset and evaluates these methods by their ability to answer domain questions.

3 Data

3.1 Data Collection

Our dataset is from GISAID, which stands for Global Initiative on Sharing All Influenza Data [2]. As the name implies, their primary focus is Influenza data, but in 2020, they shifted to COVID-19 related data. Their website has a public dataset of approximately 27000 viral genomes from patients around the world. These genomes have DNA sequences that are 30K bases long. It also has amino acid sequences that correspond to certain peptides that are encoded for in the DNA sequence. We use both the spike protein sequences and DNA sequences in our work. There are many deficiencies in this dataset, and a clustering algorithm will need to account for them.

3.2 Data Characteristics

Varying Quality And Time Periods

The data was generated from DNA sequencing machines that have a varying level of accuracy and quality score. For example, some of the data was generated from a MinION sequencing machine, which is a portable DNA sequencer that has approximately 96% sequencing accuracy. Other samples were generated from an Illumina sequencer which has close to 99.99% sequencing accuracy.

In addition, the sequences were collected over a range of time. There are different numbers of sequences sampled at different times, and these sequences can demonstrate significant variance from mutations, lab biases, sequencing error, lab protocols, and country of origin. More variance often means clustering will be more difficult, because the results might pick up on confounding variables rather than true trends. We can see the distribution of the number of sequences over timestamps below.

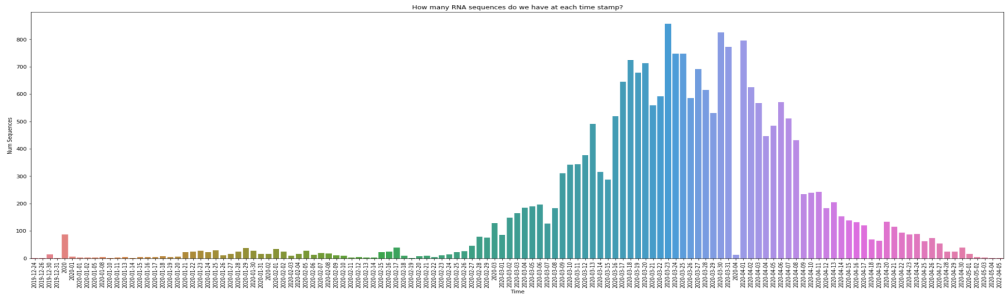

Figure 1: Distribution of sequences across timestamps

Varying Levels of Specificity

DNA sequences typically contain nucleotides ‘A’, ‘T’, ‘G’, and ‘C’. However, when a sequencing machine is not confident about the base, it will output an ‘N’. Most sequences in the GISAID dataset have long sequences of ‘N’s at the beginning and end of the sequence, which varies the amount of information that is encoded in a particular sample.

Example sequence: "NNNNNNNNN...CTTGT ...NNNNNNNNNNNNNN"

Varying Length of DNA Sequences

The majority of DNA sequences in the database are approximately 30K nucleotide bases long, but lengths vary. It is likely that a sequence significantly shorter than 30k bases long is the result of a faulty machine or is in some way erroneous. Viral genomes are generally 30k bases long, and it will be important for us to do some sort of quality filtering when developing a clustering algorithm.

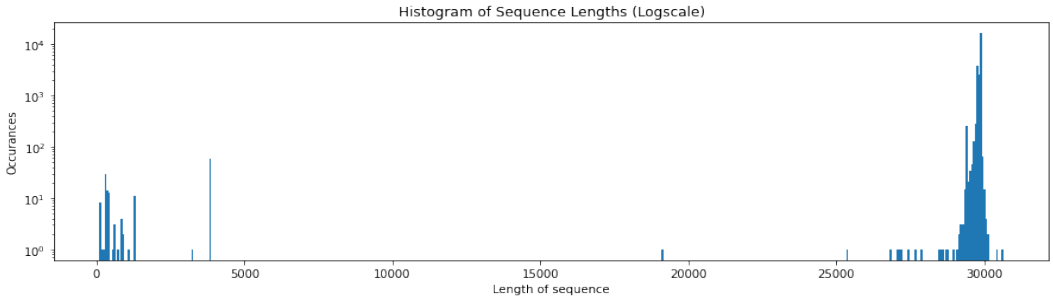

Figure 2: Distribution of sequences lengths

Size of the dataset

The dataset has data for approximately 27000 patients. While this is not a dataset as large as some of the datasets we have seen in the class, each of the individual sequences are rather long, and this will still present challenges in computational complexity.

Spike Protein Dataset

Around May, GISAID uploaded a dataset of just spike protein samples for the majority of patients in their database. The spike protein sequence is encoded for in a portion of the full viral genome. This spike protein sequence is around 1300 amino acids long. Amino acids make up protein sequences. We can see that the number of peptide sequences varies over time in a similar manner to the DNA sequences:

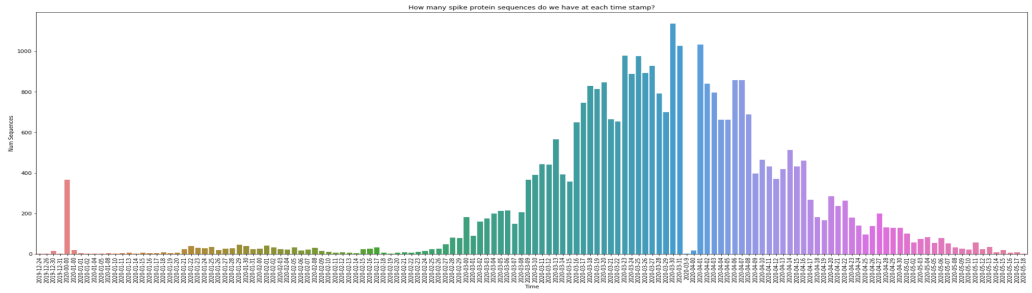

Figure 3: Distribution of spike protein sequences across timestamps

We ultimately use this dataset of spike proteins to make our clustering algorithms useful. This works better because these sequences are much shorter than the DNA sequences and allow our algorithms to converge in a reasonable amount of time.

4 Domain Questions

We have two basic domain questions about our dataset:

1. Are sequences more similar based off of time or region?
2. How does Coronavirus evolve over time in a city? Can we detect multiple strains of the virus?

In order to answer these questions, we propose two experiments. In one experiment, we pick data from numerous cities with a lot of data and a wide range of time values. We can apply clustering and see if the resulting clusters are more likely to be grouped by location (i.e. city), or by time. The second experiment is to pick one city with a lot of data and a wide range of time values. With clustering, we can see if we can generate clusters that are across distinct time periods. This means that a couple of consecutive weeks are clustered together. By consulting with travel patterns and media coverage, we can see if these particular time based clusters correspond to the presence of multiple strains. If the clusters are grouped based off of time, it is also likely that they indicate how the virus has progressed in that city.

5 Algorithms

This section details the seven algorithms we applied, or attempted to apply to this dataset.

5.1 Starcode

Starcode uses a trie structure to dynamically calculate Levenshtein edit distance between sequences. Typically represented as a dynamic programming problem using a matrix, the trie structure allows shared prefixes to be exploited to represent multiple string comparisons (instead of a matrix per pair). Every time one DNA sequence entry is added into the trie, it partitions the data into indexes and reduce the length of each segment. During the clustering phase, the model sorts the input to maximize prefix sharing and store the intermediate matching results in the trie node. We found and used a Starcode implementation on Github [7].

5.2 MeSHClust

MeSHClust uses mean shifting with a classifier to discover clusters. Each sequence is represented as a histogram of common predetermined k-length strings. Starting with a random sequence as the centroid, the mean of all “similar" sequences to the centroid is found. The sequence most similar to the mean is the new centroid. The process is repeated until the centroid does not shift. That centroid and its “similar" sequences are defined as a cluster. This is then repeated with a new sequence clustering the remaining sequences, until every sequence is assigned. Similarity is defined using a binary classifier trained on global alignment of sequences where a similarity threshold is user defined. We found and used a MeSHClust implementation on Github [8].

5.3 Clustering Billions

Clustering Billions starts by synthetically creating sequences by making noise copies of existing sequences. Noisy copies are made by keeping, deleting, replacing, or duplicating each character in a sequence based on a distribution. Each sequence is then hashed across computational nodes, where they are clustered by q-distance which is Hamming distance where indicators are the existence of q-gram nucleotide subsequences. Centroids are chosen by randomly sampling from a cluster. Clusters whose distance are below a threshold θ are combined. The distance function of two clusters is:

$$C_1, C_2 \in \tilde{C}, d_E(C_1, C_2) = \min_{x \in C_1, y \in C_2} d_E(x, y)$$

where d_E(x,y) is the edit distance between the two sequences. The centroids are then rehashed across computational nodes and repeated.

For this project a custom version of Clustering Billions was implemented. We elaborate in Section 5.5.2.

5.4 Fuzzy Integral Similarity

This algorithm generates a $N * N$ distance matrix between N sequences by iterative defining the k^{th} distance matrix until convergence between the k^{th} and $k + 1^{st}$ matrices. Convergence is defined with respects to the root mean square distance between the distance matrices:

$$RMSD(D^k, D^{k+1}) = \sqrt{\frac{\sum_{ij \in D} ||D^k[i,j] - D^{k+1}[i,j]]||^2}{n}} < \epsilon = 5 * 10^{-7}$$

where episilon is defined in the paper. Each index of the distance matrix D is defined a:

$$D[i,j] = D[j,i] = 1 - FISIM(P_i^k, P_j^k)$$

where P_n^k is the k^{th} Markov chain transition probability matrix of nucleotide pairs of the n^{th} sequence. The k^{th} Markov chain is simply the k^{th} power in terms of matrix products of the base transition matrix. FISIM is the fuzzy integral similarity between two Markov transition matrices:

$$FISIM(P_a^k, P_b^k) = \max_i^{BASES} \max_j^{BASES} \min[h(y_{ij}), \mu(A_{ij})]$$

where BASES is the number of unique nucleotide in the sequence (paper uses a constant 4), $h(y_{ij})$ is $1 - |P_a^k[i,j] - P_b^k[i,j]|$, $A_{ij} = y_{i1}...y_{ij}$ where y_{ij} is the max of the ij-th element between the input matrices, and $\mu(A_{ij})$ is defined through the union of element sets. Let these sets be X, Y:

$$\mu(X \cup Y) = \mu(X) + \mu(Y) + \lambda_i \mu(X)\mu(Y), \forall \lambda_i$$

where $\mu(y_{ij}) = y_{ij}$ and λ_i is defined in:

$$\lambda_i + 1 = \prod_{j=1}^{BASES} (1 + \lambda_i y_{ij})$$

Note, that while the algorithm itself is computationally complex, it is not affected by the length of a particular sequence since each sequence is represented by a transition probability matrix, which is a square matrix of length equal to the number of possible bases in a sequence. Once the distance matrix is obtained, it can then be used as part of a clustering algorithm.

For this project a custom version of this algorithm was implemented and we elaborate in Section 5.5.3.

5.5 Custom

The algorithms we applied to this dataset purely from background research are not tailored to our dataset. As stated earlier, our dataset is fairly low quality and has a lot of variance. Furthermore, the sequences are very long, and a traditional clustering algorithm is unlikely to converge, especially since they are likely to use sequence alignment and edit distance. Based on our experiences, we decided that we should edit the existing algorithms in an attempt to make them work better on our dataset.

5.5.1 Custom K Means

Our first attempt at a custom clustering algorithm was a modified K Means algorithm. K Means is not particularly well suited for this problem, but it gives us insight into the complexity of this dataset. The goal of K Means is to minimize the distance in a cluster and maximize the distance between clusters. Given that the sequences are non-numeric, we used edit distance to compare sequences and assign them to a cluster. In traditional K Means, a new cluster center is picked based off the "mean" of the sequences that were assigned to the cluster. Since, it is not possible to take the mean of strings in a traditional sense, a random DNA sequence in a cluster was sampled represent the new centroid.

5.5.2 Custom Billion Reads

This algorithm was initially designed to run on a multi-core distributed system for parallel speed up. However, since the GISAID dataset that we are working with is smaller than the paper's input by magnitudes, we create a variation of the original algorithm that computes and updates locally.

We start with modifying the agglomerative implementation from the Billion Read paper on DNA sequences. Since we are only running the algorithm on a single-core machine, we took out the global hashing and redistribution steps to make it suitable for our task.We explored two distance functions to measure the similarity of clusters: 1. $d_{edit}(seq1, seq2)$ compares two sequences as strings and returns the edit distance of the two. 2. $d_{align}(seq1, seq2)$ uses the Biopython pairwise aligner with reward of 1 and gap penality of -5. We determine the distance of two clusters with $D(C_1, C_2) = \max(d(seq1, seq2))$.

Algorithm 1: Custom Billion Reads

Function `cluster(`*S, iterations,* θ`):`
- $\tilde{C} \longleftarrow S$;
- **for** $i = 1, 2...iterations$ **do**
  - **for** $C_x, C_y \in \tilde{C}$ **do**
    - $dist \longleftarrow D(C_x, C_y)$
    - **if** $dist < \theta$ **then**
      - Update $\tilde{C} = (\tilde{C} \backslash \{C_x, C_y\}) \cup \{C_x, C_y\}$
    - **end**
  - **end**
- **end**
- **return** S^*;

End Function

5.5.3 Custom Fuzzy Integral Similarity

Recall that each sequence is represented as a transition matrix between nucleotide bases (The choice of presenting using BASES instead of the paper's "4" is to emphasize this point.). Updating the algorithm to scale with the different nucleotides and sequence lengths simply involved changing the shape of the matrices.

6 Results

6.1 Evaluation Criteria

We evaluate our algorithms based on their ability to answer the broad domain questions we discussed earlier. From intuition, it seems as if sequences that were collected around the same time or in the same location will be similar. This is because sequences in the same time frame are likely to have similar mutations, and sequences from the same location may be similar because of equipment used or similar lab protocols. An ideal algorithms will allow us to answer both of these questions effectively.

6.1.1 Moving from clustering DNA to clustering amino acid sequences

At the beginning of this project, we were working with the DNA viral genome dataset. As we began to apply and implement algorithms, we realized that there would be a significant computational

bottleneck because of the cost of alignment and edit distance calculations. Around this point, we found that GISAID had released a spike protein peptide dataset that was about as large as the DNA dataset. The spike protein is one of the most important proteins encoded for in the genome. Labs are currently doing research into spike protein based vaccines. These sequences are amino acid sequences, so the alphabet is now 20 characters, not 4 characters like DNA sequences. In this section we present results for both of these datasets and use the protein dataset to answer domain questions because we were able to tune our algorithms only on this dataset.

6.2 Starcode

The implementation of Starcode found restricts sequences to only the characters 'A', 'T', 'G', and 'C' and to less than 1023 bases. There are only about 25 sequences that meet this criteria. Starcode clustered the 25 sequences with a Levenshtein distance of 8, and found 25 clusters, which are all singleton clusters (i.e. only one sequence in them). If a viral genome sequence is significantly less than 30k bases long, it likely broke during sequencing meaning the output from a sequencer is not particularly meaningful. This in combination with the fact that Starcode could not cluster different sequences into the same cluster makes our Starcode results not useful.

6.3 MeSHClust

The implementation of MeSHClust found restricts the input characters to 'A', 'T', 'G', and 'C', which reduces the available sequences to approximately 4500. Experimenting with a variety of k-mer values results in MeSHClust assigning each sequence into its own cluster regardless of parameter tuning. In general, the MeSHClust algorithm touches on one of the big challenges when comparing DNA sequences that are very long; it is hard to encourage clusters to include more than one sequence. MeSHClust in particular is not appropriate for our dataset because it cannot account for sequences that have an 'N', and because regardless of what hyperparameters we choose, the clusters only contain one sequence in them. Like Starcode, the MeSHClust results are not meaningful.

6.4 Clustering Billions

The original unmodified algorithm is by definition unsuitable for our task as the original paper's goal was to cluster synthetically generated versions of the same sequence. This project is to explore the relationships between different viral genome sequences. Section 6.6.2, will discuss the results of the modified algorithm.

6.5 Fuzzy Integral Similarity

Similar to MeSHClust, the original Fuzzy Integral Similarity algorithm restricts the input characters to 'A', 'T', 'G', or 'C'. Attempts to generate distance matrices for these sequences failed to converge. The process and findings are explained in more detail with the custom version of this algorithm in Section 6.6.3.

6.6 Custom

6.6.1 Custom K Means

We chose 10 randomly selected sequences as the initial clusters. The initial choice of 10 clusters was an arbitrary starting point. Recall that we also chose new clusters randomly because we can't take the mean of strings. Overall, K-means when applied to the whole dataset is computationally very expensive because at the minimum, we had to compute pairwise edit distances between every sequence in our dataset. This would take weeks without parallelization, so we demonstrated the results on a small subset of our data. The following graph describes how the cost of the clusters based on edit distance changes over 20 iterations:

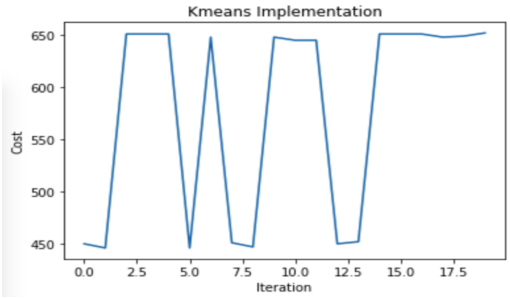

Figure 4: Kmeans Cost v. iterations

In the above graph, the cost fluctuates unpredictably because we select new cluster centers randomly. In general, these results are not meaningful because K-means is not well adapted for DNA sequences. An algorithm like K Means will not work for this domain because it relies on pairwise comparisons which are very expensive for sequences that are of length 30k. An algorithm could potentially use parallelization though, to reduce the cost of computing pairwise alignment scores, or not use alignment/edit distance at all to compute similarity. Attempts with parallelism, different initial centroids, and different number of clusters were made, but did not produce meaningful results in a reasonable amount of time. Given this, we choose to spend our time working on algorithms that are more suited to DNA and protein sequences.

6.6.2 Custom Clustering Billions

We ran the algorithm on the two selected datasets mentioned in the data section to test whether Billion reads can pick up features like time or region to determine the clusters. To speed up the algorithm, we pre-computed the pairwise distances of the input data and feed into the clustering algorithm. Initially, we attempted to compute pairwise DNA alignment score. However, the average length of the nucleotide per sequence is around 20k, making it impossible to finish the alignment computation in a reasonable amount of time. In addition, given the complicated nature of the aligning DNA sequences, we cannot divide the original sequences to speed up the computation.

Our attempt using spike protein as input was fairly successful. The average length of peptide sequences are 2.5k per data point and we were able to compute distances between pairs of spike protein records. We run our customized Billion Read algorithm on the precomputed distance function output and got back clustering results based on time progression and region of origin.

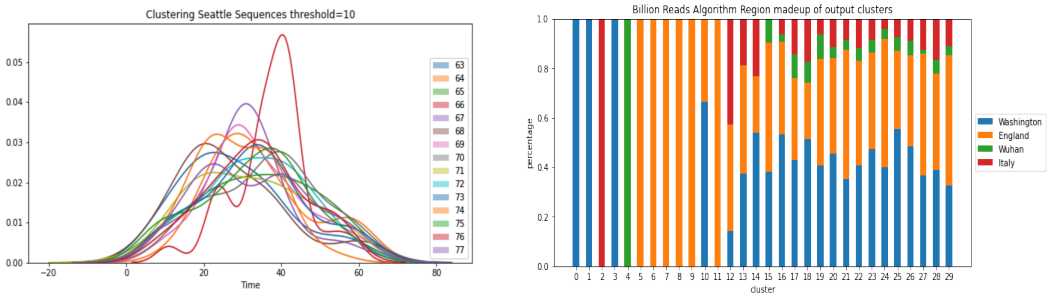

Figure 5 (Left) is the clustering output of all the data points collected from Seattle area. We collect the timestamp of each data point and plotted the distribution. Each color represents a distinct cluster from the output of the algorithm. Figure 6 (Right) uses a input dataset that contains spike protein samples from four regions across the world: Washington state, England, Wuhan and Italy. Each vertical bar represents a cluster from the output of the algorithm. Different colors marks the different origin of the data sample.

Hyperparameter - The algorithm is guaranteed to converge regardless of the threshold choice. However, to find the hyperparameter that works the best for the dataset, we focuses on two different quality of the output clusters. Firstly, we want to keep the total number of clusters relatively small. Since we start with number of data points many of clusters, we try to search for outputs that contains

less than 30 for region-based clustering and 100 for timestamp-based clustering. Secondly, we prefer parameters that give more even distributions of clusters. Since we synthesized the data set to be uniform across time and region, we expect all output clusters contain relatively same amount of data points.

6.6.3 Custom Fuzzy Integral Similarity

The output of Fuzzy Integral Similarity generates a distance matrix between all input sequences. Applying the algorithm on the DNA dataset was initially unfruitful; the model was unable to converge when the number of sequences was greater than two. Correspondence with the original authors of the paper revealed (after some confrontation) that there were numerous errors in the algorithm provided in the original paper. Three of the errors in the algorithm of the original paper include incorrect termination conditions, inconsistent max/min domains, and undocumented loop encapsulation (Note the algorithm described earlier, as verbose as it is, is actually a simplified version of the fixed algorithm).

After accounting for the discovered errors, the algorithm converges for smaller samples of the sequences, but still has trouble converging with larger data sets and larger counts. Therefore, similar to Custom Clustering Billions, we cluster the spike protein data for Seattle and plot with respects to the timestamps.

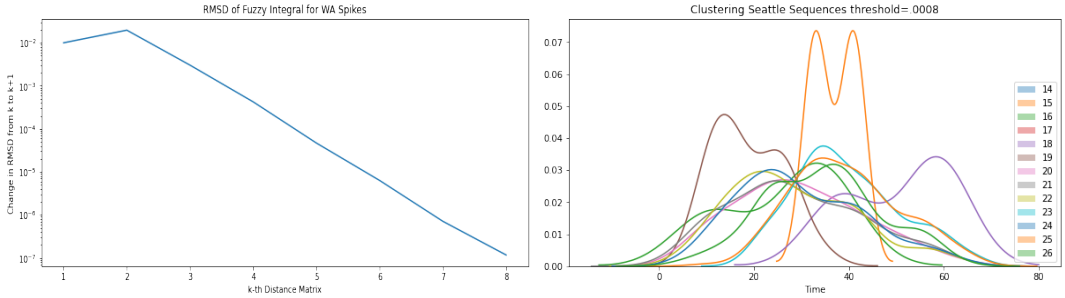

Figure 7 (Left): RMSD between the K and K+1 distance matrix for sequences from Washington in the Spike dataset. Convergence of the algorithm occurs on the 8th degree Markov Matrix. Figure 8 (Right): Histogram of timestamp distributions of clusters sequences using the K=8th distance matrix.

Hyperparameter - The choice of threshold was found by iteratively testing using a step size of .0001 from .0001 to .001, with .0008 being the only threshold that did not result in either uniformly distributed clusters (every comparison results in a cluster merge) or all singleton clusters. Similar to before, the number of clusters chosen is the point when no clusters were merged over 3 iterations.

7 Analysis

7.1 Custom Clustering Billions + Custom Fuzzy Integral Similarity

We demonstrate the results of our clustering algorithms using histograms. We only chose to plot the non singleton sequence clusters in both Fuzzy Integral and Clustering Billions. The x-axis is time, and the y-axis is density. According to Figure 8, Fuzzy Integral Similarity groups together consecutive timestamps fairly well. It also has a wider diversity of curves than we see in the corresponding histogram in Figure 5 for Clustering Billions. It seems that Fuzzy Integral Similarity is more inclined to cluster by time.

We plot the clusters across four regions using the Clustering Billions algorithm in a stacked bar plot in Figure 6. The x-axis represents clusters and the different colors correspond to different regions. We see that many of the clusters have data from only one of the four regions. It seems like the Clustering Billions algorithm is more likely to group elements based off of region.

7.1.1 Answering Domain Questions: Seattle

It seems like Fuzzy Integral Similarity clusters sequences from Seattle more based off of time than region. Since the clusters that Clustering Billions generates from the Seattle sequences mostly span

across all of the timestamps, it seems like it clusters sequences more based off of location. The Fuzzy Integral Similarity algorithm is more useful in Seattle, because there is only one region. Notably, news outlets and media sources suggest that there have been three strains of Coronavirus in Seattle, and we see three clusters (19, 15, 18) in Figure 8 that seem to represent this. This means that we can see the progression of the disease in Seattle with Fuzzy Integral Similarity. For the Seattle sequences, it seems like Fuzzy Integral Similarity most effectively answers the domain questions.

7.1.2 Answering Domain Questions: Wuhan, Washington, England, Italy

Our second use case was using data from four major disease epicenters. According to Figure 6, Clustering Billions seems to fairly effectively cluster based off of the location of origin of the sequence. Many of the clusters contain only sequences from one region, and the clusters that have multiple regions seem to clearly have a region that has a majority of the sequences for that cluster. However, we can't detect progression over time for these clusters, or any sort of mutation. This means that Clustering Billions was able to answer one of the domain questions, and that it is not necessarily good at answering the other one.

7.2 Challenges + Difficulties

This was an exploratory project, and we spent a lot of our time accounting for the low quality in our dataset and computational challenges associated with the nature of these sequences.

Clustering algorithms on sequences with length > 1000 takes a long time. The best ways to compare two DNA sequences are through edit distance and sequence alignment. However, both edit distance and alignment become are very computationally expensive, and have an $O(n^2)$ complexity. While this is still okay for shorter sequences, it becomes problematic for longer ones. If we were to use existing algorithms without modifying them, the algorithms would take months to converge. By shifting to using only spike protein amino acid sequences, we can reduce this problem into a more manageable one.

In general, traditional algorithms are very restrictive of the input sequences. Clustering algorithms for DNA sequences are often very specific to certain tasks. As such, it is tough to generalize them to work on other datasets, especially ones that have low quality and poor standardization. When we decided to shift to clustering spike protein peptide sequences, we found that only two of the existing algorithms could be modified to work for our problem.

8 Conclusion

Our clustering results generally show that we can group sequences by regions and by time. Neither of our clustering algorithms were able to effectively do both, and it often depended on the spread of our data. Regardless, the Fuzzy Integral Similar algorithm is better for grouping by time, and the Billion Reads algorithm is better for grouping by region. Both of these algorithms when used in conjunction, can effectively answer our domain questions.

Additionally, the spike protein is an effective subsection of the viral genome to cluster on. We had to reduce our problem to only consider the spike protein because neither of these algorithms converged with sequences that were very long (i.e. 28K + bases). However, the spike protein carries enough information about the sequence to allow us to answer these domain questions with our algorithms.

Future work on this project includes parallelizing our custom algorithms, and applying this clustering pipeline to additional data from other cities and other protein sequences in the COVID-19 genome. While we found that we can effectively reduce our problem to just analyzing the spike protein sequence, we believe there are meaningful results that can be obtained from other proteins. Furthermore, we can analyze other cities, that, for example, have a larger number of labs in them or have more data. England has the largest number of sequences in this database, and it is likely that we can better detect mutations and travel patterns through those sequences.

References

[1] "Provisional Death Counts for Coronavirus Disease (COVID-19)," Provisional Death Counts for Coronavirus Disease (COVID-19) | HealthData.gov, 06-May-2020. [Online]. Available: https://healthdata.gov/dataset/provisional-death-counts-coronavirus-disease-covid-19.

[2] "Initiative," GISAID. [Online]. Available: https://www.gisaid.org/.

[3] Eduard Zorita, Pol Cuscó, Guillaume J. Filion, Starcode: sequence clustering based on all-pairs search, Bioinformatics, Volume 31, Issue 12, 15 June 2015, Pages 1913–1919, https://doi.org/10.1093/bioinformatics/btv053

[4] Benjamin T James, Brian B Luczak, Hani Z Girgis, MeShClust: an intelligent tool for clustering DNA sequences, Nucleic Acids Research, Volume 46, Issue 14, 21 August 2018, Page e83, https://doi.org/10.1093/nar/gky315

[5] Cyrus Rashtchian, Konstantin Makarychev, Miklós Rácz, Siena Dumas Ang, Djordje Jevdjic, Sergey Yekhanin, Luis Ceze, and Karin Strauss. 2017. Clustering billions of reads for DNA data storage. In Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17). Curran Associates Inc., Red Hook, NY, USA, 3362–3373.

[6] Saw, A.K., Raj, G., Das, M. et al. Alignment-free method for DNA sequence clustering using Fuzzy integral similarity. Sci Rep 9, 3753 (2019). https://doi.org/10.1038/s41598-019-40452-6

[7] Gui11aume, "gui11aume/starcode," GitHub. [Online]. Available: https://github.com/gui11aume/starcode/tree/master/src.

[8] TulsaBioinformaticsToolsmith, "TulsaBioinformaticsToolsmith/MeShClust," GitHub. [Online]. Available: https://github.com/TulsaBioinformaticsToolsmith/MeShClust/tree/master/src.