A Graph Representation of the Genomes Collected in a Viral Outbreak

Final Report

Authors: Caroline Turner, Drew Pompeii, Jake Fisher, Jared Hubert, Michele Ambrose

Client: Dr. Lenwood S. Heath

CS 4624: Multimedia, Hypertext, and Information Access

Professor: Dr. Edward A. Fox

Virginia Tech, Blacksburg, VA 24061

May 13, 2021

Table of Contents

Table of Figures

Table of Tables

I. Abstract

The best way for researchers in the biology field to understand how viruses mutate, especially during the COVID-19 pandemic, is to compare the different genomes. Our project's goal was to create a software that visualized the comparisons to support this endeavor. The software we created is meant to support the researchers working with the SARS-CoV-2 virus during the 2020 pandemic, with the goal of finding out how the virus mutates and the different strains created. Though our primary goal was to work with the SARS-Cov-2 virus, our software does support comparisons for any genome that needs to be studied.

Our methods include creating an interactive user interface (UI) where the user can upload different genomes to compare in the visualization. The user can then print or download the visualization that was created, search different genomes stored in the CSV file, and delete any unneeded genomes in the file. The research completed for this project was focused heavily on understanding how viral genomes were viewed and compared as well as learning more in the research area of bioinformatics and computational biology.

We were challenged to further our understanding of how to easily portray the genomes in an easy-to-read manner. Our research covered many computational areas, though we were tested in understanding the biology side of the project. This was a fascinating endeavor that furthered our education and opened our minds to understanding the basics of the bioinformatics field. We were able to finish a working prototype by the end of the semester and we hope to see the project grow and adapt to stay relevant in the future.

II. Introduction

Genomes are composed of a DNA sequence. They are analyzed by comparing the DNA base arrangement of the genome. These bases are adenine (A), guanine (G), cytosine (C), and thymine (T). The different arrangements of these bases create the genetic code of every living being.

A mutation is defined by the National Institutes of Health (NIH) as a change in the DNA sequence [1], which will then change the resulting genome into either a different gene trait, a different breed of the species or even an entirely new species. A mutation can be as small as a single change to the DNA sequence.

Virus genomes often are analyzed to see the different mutations that create each strain of each disease. These analyses are then used for further research on different treatments and vaccines needed to keep the human population safe from the diseases.

Over the course of the COVID-19 pandemic, many bioinformatic scientists have obtained many different samples of the SARS-CoV-2 virus. Using DNA sequencing technology, these samples have been analyzed to find that many of the different strains of this virus have differed by a single mutation.

To better understand the nuanced changes of these sequences, our team has created a visualization tool to help see how each strain is related to each other. This tool will be able to compare and contrast different genomes to see how many mutations there are in between the genomes, and where they occur.

Though our tool is geared towards working with viral genomes, it can also work to visualize comparisons of other types of data.

III. Requirements

The specifications of this project were given to us by Dr. Lenwood Heath. We were tasked to create a visualization tool to help show the differences between different genome strains, specifically viral genomes. Our main focus on the project was on the user interface and how the genomes would be visualized. Our client, Dr. Heath did not have many specific requirements for this application. We were asked to be sure that it could be used for multiple genomes, as it would be the most beneficial in this use, though we were told that the highest number of genomes accessed at a time would be less than a hundred. This specification was necessary to help with our efficiency coding, as genomes are lengthy strings of letters, averaging upwards of 250 base pairs.

Our software was required to be able to read in and process the genomes from a FASTA file, which is the file type most commonly used for reading genomes. Once the FASTA file was parsed into our application, it needed to be able to align and trim the different genomes, and then would be able to show in our graph the different mutations from the genomes given.

IV. Design

Our design for this project was originally to create a web application that could be accessed easily. This design plan was diverted to a stand-alone application because our team found that would be an easier method to create the project environment and also to maintain the application for future upkeep. Our client, Dr. Heath did not have a personal preference as to the type of application.

Our next step in our design process was to understand what the different parts of the application would do and how it would work. The backend of the application would focus on file parsing, storage, and handling the alignment of the genomes. The frontend would focus on creating the graphs from the data given in CSV file format. The properly aligned genomes would then be shown in the user interface as a graph representation of the genomes' mutations.

V. Implementation

Frontend

Graphs

The graph will be the main interaction point for the user. It will represent the genomes as nodes and the mutations of the genome as edges between mutated genomes. This is the main element of the visualization aspect of this project. We want the graph to be easily understandable as well as easy to manipulate so that the user can better understand the mutational data. Our graphs and user interface were adapted and modified based off of supporting code found in graphical sources found online [1]. We then took this knowledge to implement using the plotly, networkx, pandas, and dash packages in Python, to give us our own unique graphical interface to represent SARS-CoV-2 data.

User Interface

The graph was designed to be interacted easily with, so the user can view the data as they wish. At the top of the graph (as shown in Figure 1) is a bar designed for the user interface. This bar has a series of icons that allow the user to zoom in and out, select a single part of the graph, compare the data between two different genomes, and save an image of the graph to their computer. Additionally, when you hover over a node on the graph, it will display information about that specific genome such as its identifier and the type of mutation. When you click on a node, information will be displayed to the right of the graph containing more detailed information about that genome [2]. This information will include genome information such as its identifier and mutation, but it will also give details such as where the node is located on the graph, the date of the genome mutation, and how many other nodes it is connected to. This is to provide easier analysis for the user.

***Figure 1**: Bar at the top of the graph that helps with graph user interaction*

Mid-Application Data Manipulation

The processes that fall into this category were designed to be run at will while the user is analyzing genomes to further filter down the genomes that they want to look at. Because we do not want to consistently slow down during the application when we can avoid it, efficiency of code was of the utmost importance. We also wanted as many methods as possible to be somewhat generic, allowing us to perform many of these methods on the different types that are/possibly included in the datasets. The

different types of processes in this category can be broken up as such: parsing, sorting, searching, and range.

These methods allow the user to specify the data by a range of collection dates, a collection location, a minimum length, or a variety of all three. They also allow the user to completely change the data that is being viewed by choosing different genome and edge files. The user can also export the data to their current directory with these methods.

Parsing

The main job of these functions is to import a given file, whether it be from the user or passed to the program from the back-end, then scan that file to determine the genomes and their characteristics. Because we will be using .csv files, the data is particularly easy to parse due to everything in the file being comma-separated. All of the genomes in this phase are stored within a 2 dimensional list, so access to the elements is simple and efficient.

The same information applies to the creation of the edge list. It is also a 2 dimensional list, but instead of parsing a file, we are instead parsing the genome list. Due to the nature of calculating every possible edge and the number of differences in the corresponding genomes, this algorithm unfortunately takes $O(n^3)$ time. This isn't the end of the world though since this algorithm will only be run once for every different .csv file that is included by the user.

Sorting

The purpose of these algorithms is to, given any 2 dimensional list, and a category index, sort that list based on the specified category. The sorting algorithm that we are using for this project is the mergesort algorithm. The mergesort algorithm was chosen partially because of its relative simplicity when compared to quicksort, but mostly due to its efficiency being O(nlogn). We needed an efficient algorithm more than a simple algorithm because the user may choose to sort their data by many different categories throughout their analysis.

Searching

These functions are meant to look for any data that begin, contain, or end with any particular string. All of these functions are O(n) time since we only need to search one category of every element in our genome or edge datasets. The searching is done via exact matching for every specified category. These methods are not heavily used in the final product and are most useful when the user is trying to be very specific. For example, these methods could be useful if the user has multiple differently named types of genome in their analysis, but the chances of this are not particularly high. The elements that satisfy the search function are returned as a 2 dimensional list to be saved.

Range

This set of functions were designed so that the user could select all genomes or edges that have some sort of data category that falls into a specified range. These functions, much like searching, are O(n) time. These functions look at the specified element of every genome/edge and check if the element falls

within or outside a range of 1 (for greater/less than ranges) or 2 (for including/excluding ranges) specified values. The elements that are within the specified range are returned as a 2 dimensional list to be saved.

Backend

Our first approach to the backend was to build a database that would hold the genomes being compared. We started with research into which database platform to work with and were starting work on a PostgresSQL database. We ended up aborting this approach when the front end was further implemented since we found it was easier to just use a comma separated value (CSV) file stored locally instead of building a database and running a server, which could become costly. We talked to our client, Dr. Heath, and he was amenable to the change. Because of this change, Dr. Heath suggested the use of FASTA files, as these are the usual format genomes are stored in. Since these were a type of text file, it was a simple concept to implement, and creates less converting for our tool or the user.

Our implementation uses Python v3.6 and made use of the BioPython library [3], which is an open-source collection of tools for computational biology and bioinformatics. We found this library made it simple to parse the FASTA files [4] and be able to acquire the names and sequences of the different genomes in the FASTA file, as each file usually contains multiple genomes. This library also contains many different methods to help manipulate and manage the genome data.

Once the data is parsed, we store the data in a CSV file, which is titled “Fasta_input.py” in our implementation. The majority of our backend is coded in the “backend.py” file, which handles adding, removing, searching, and comparing genomes in the tool.

Acquiring Genomes

Our implementation of acquiring genomes takes the FASTA file provided by the user and parses it in the backend [5]. The genome sequence is then trimmed and aligned using the algorithm used in the BLAST program [6], which creates the most efficient possible path of the differing genomes and finds the best way to align them.

VI. Testing/Evaluation/Assessment

We've currently only used local testing in our development process. The machines our team used for the testing phase were run with Windows 10 operating systems. The tests involved running the application and trial and error on how the visualizations looked as well as comparing the graph data to the data produced by the CSVs. We manually checked the data in each node to make sure the collection date, location, and isolate name matched those in the original data. Continuing this process, we tested the data manipulation within the graph by comparing the returned data with the sorted data from the original files matching the parameters that were entered.

After getting the functionality of the user interface working, we moved to testing the back end. We tested the back end through the BioPython output the program was producing. We were able to check the output for clarity in genome sequences as well as length.

VII. User's Manual

There are some requirements needed to properly run this code. The files from this project such as app.py, Eagles.csv, Covid-Base2.csv, and genomic.fna all need to be in the same directory. Additionally, the computer it is being run on must have certain installations. It must have python3, dash, pandas, and networkX installed. To do this, go from the command prompt and type "pip install ____" where the underlined part can be replaced by the desired installation package. This should work on any Operating System so long as it has a command prompt/terminal and has the files and package installations required above. Once these are installed, you can run the program by entering "python app.py" in the terminal at the directory where the files are stored. Then, click on the link provided in the terminal to open the application in the browser.

Once running "python app.py" and clicking on the link provided, it will take you to the main page of the application, as depicted in Figure 2. Once you open the application, you will see the screen in Figure 2. This main screen displays the graph and genome sequence information for the user to analyze and study. On this screen you can interact with each genome. Each genome is represented by individual nodes on the graph. When you hover over each node, as shown in Figure 3, you can see the information about each genome as it and each edge of the graph shows how each genome is mutated from which genome. The colors and widths of the edges between nodes are determined randomly, in order to make the edges easier to follow and thus make the graph more readable. The directional arrows on the edges show which the genome mutated from, and the data in the node shows how it has mutated.

To the right of the graph is additional data on the hovered and clicked on genomes so that you can compare their data. This is shown on the right side of Figure 2. It shows the nucleotide name, length, collection date, collection location, and isolate name.

Figure 4 shows the section for users to edit and update the genomes they want to study. In this section, you will be able to upload the genome file as well as the edges file for the graph. This section is reachable by scrolling down on the page.

This section also has a 'Export Data to CSV' button, which allows you to export the graph and information created in the application to a series of CSV files, which are then automatically placed in a zip folder and placed in the directory that the application is placed in.

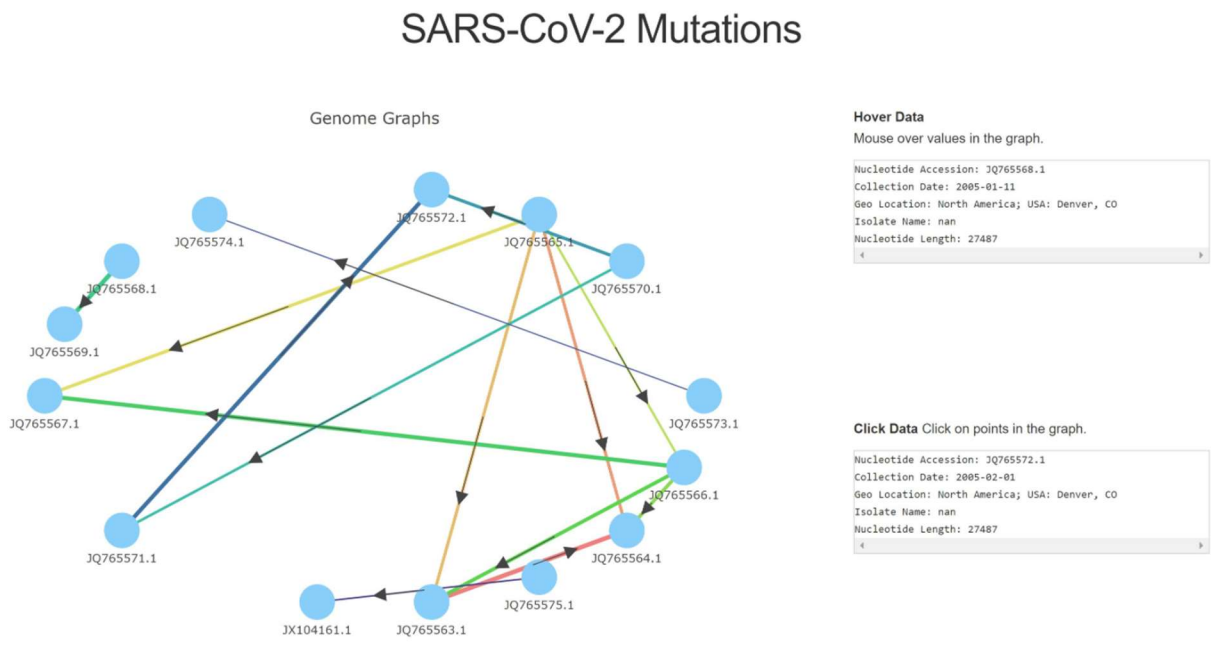

Figure 2: *Shows all the mutations of the specific genome. This is the main page of the program*

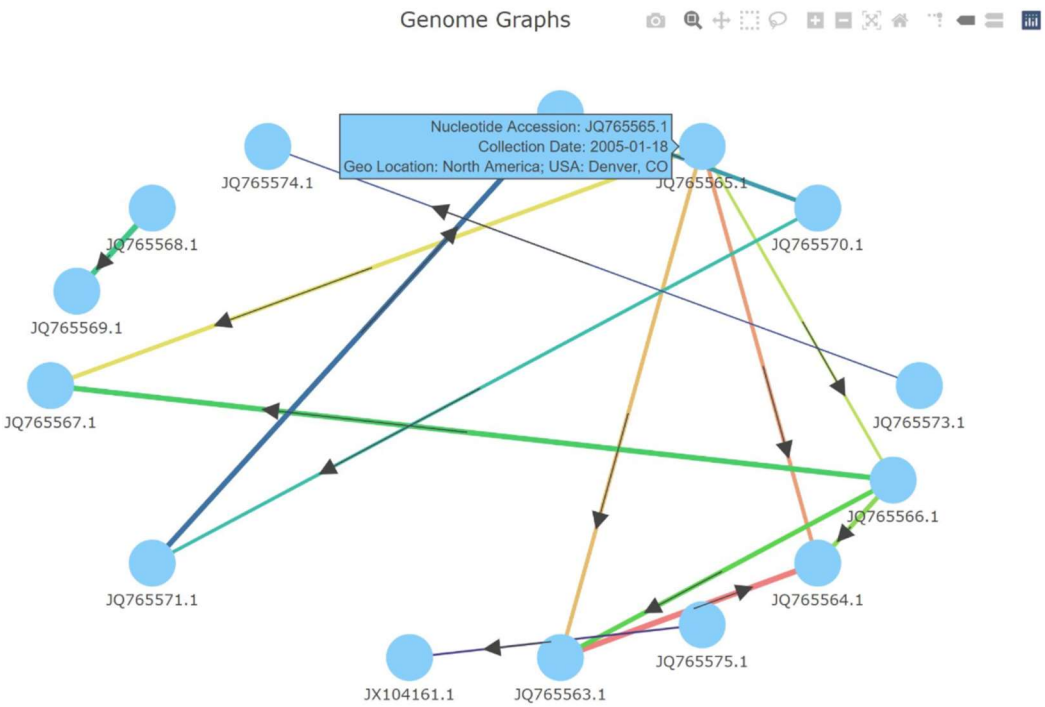

Figure 3*: Shows the different genomes as nodes and how each genome has mutated or is related to by the edges of the graph*

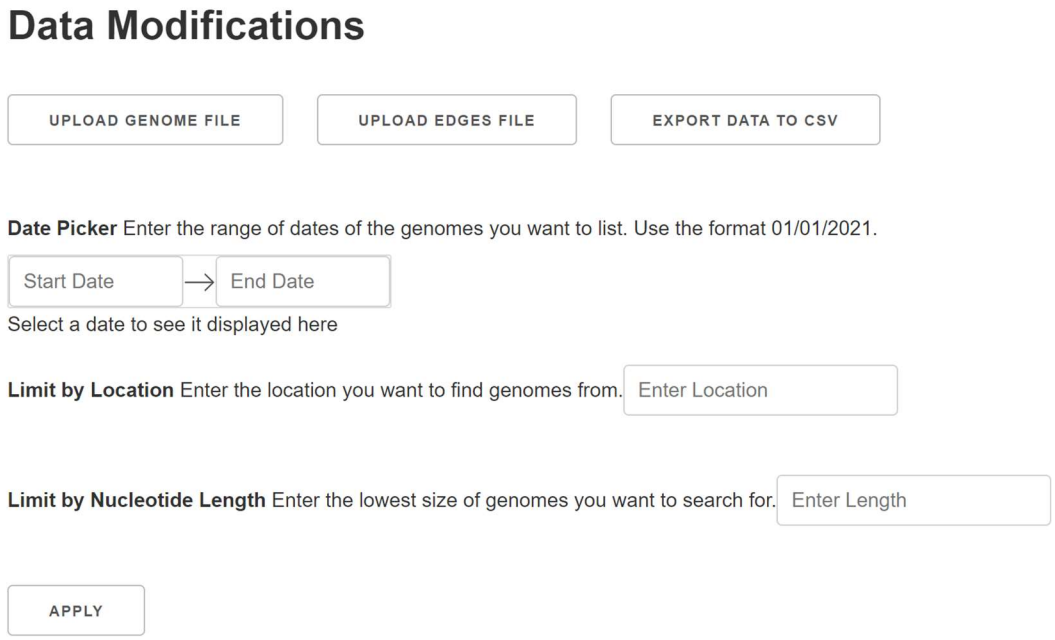

***Figure 4**: Data manipulation section. It is located at the bottom of the home screen*

VIII. Developer's Manual

In this section we will describe our methodology in our project, and how we built our code, the logic behind our decisions to go certain directions in the development process. This section's purpose is to help maintain and further the development of our application.

Methodology

The methodology is how we compartmentalized our project into the different services we will provide in our application. We used this compartmentalization to spread the workload out between our group to ensure our project was completed on time and to help design our application to be the in line with our client's requirements and expectations.

Types of Users:

Analysts - Analysts should be able to view genomes, search for genomes, compare genomes, and add new genomes to the system. They should also be able to download the current display and connections in the graph.

Maintainer - Maintainers are able to access the code and change it as necessary.

Services provided by our application:

- **View genomes** :
    1. Click and drag around the visualization and zoom in/out on the visualization to find genomes
    2. Click on genomes
    3. Show the genome information

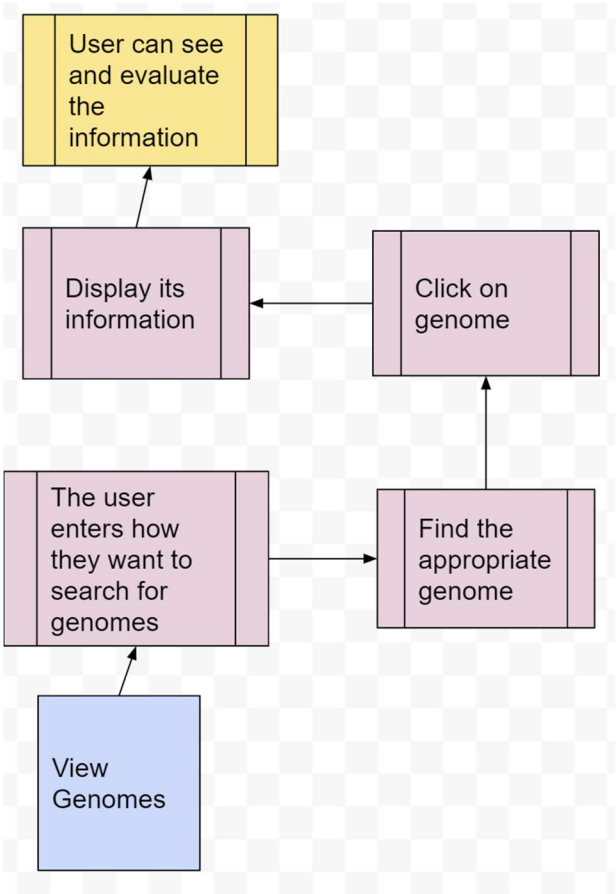

Figure 5: *Flow chart showing how the "View Genomes" service works*

- **Search for genomes:**
    1. Enter a search term
    2. Search the overarching databases for the genome/search term
    3. Return the dataset with the gathered information

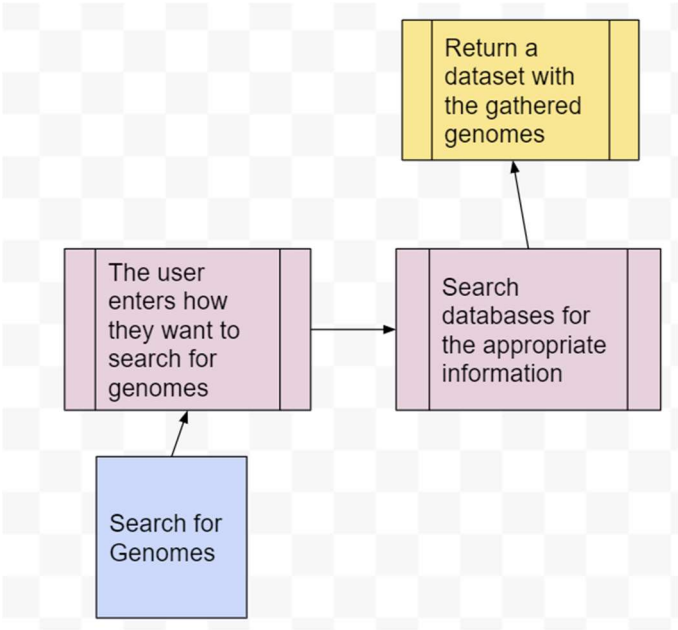

***Figure 6**: Flow chart showing how the "Search for Genomes" service works.*

- **Compare genomes**
    1. View multiple genomes at once
    2. Select a comparison method
    3. The program will perform the corresponding comparison

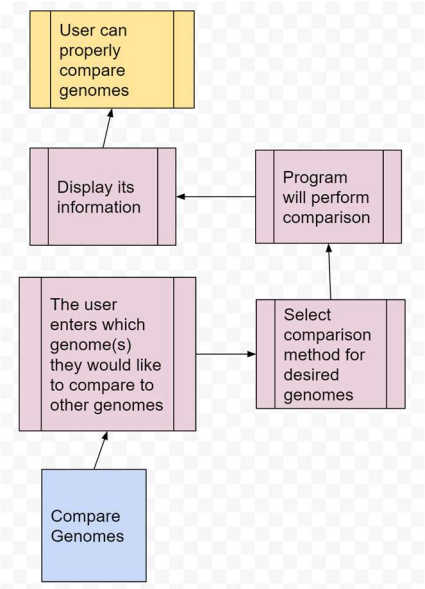

***Figure 7**: Flow chart showing how the "Compare Genomes" service works*

- **Add new genomes**
    1. Import a CSV file containing new genomes to the website
    2. Have the data imported into the current database of the user
    3. Information is sorted and edges are reallocated

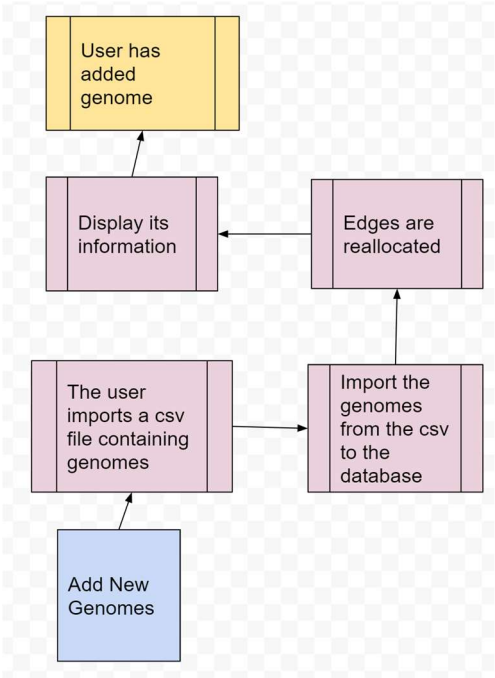

***Figure 8**: Flow chart showing how the "Add New Genomes" service works*

- **Download display**
    1. View genomes and their connections by mutation on the graph
    2. Be able to interact with the graph and better view the nodes and edges
    3. Save the graph and list of nodes and edges to their computer.

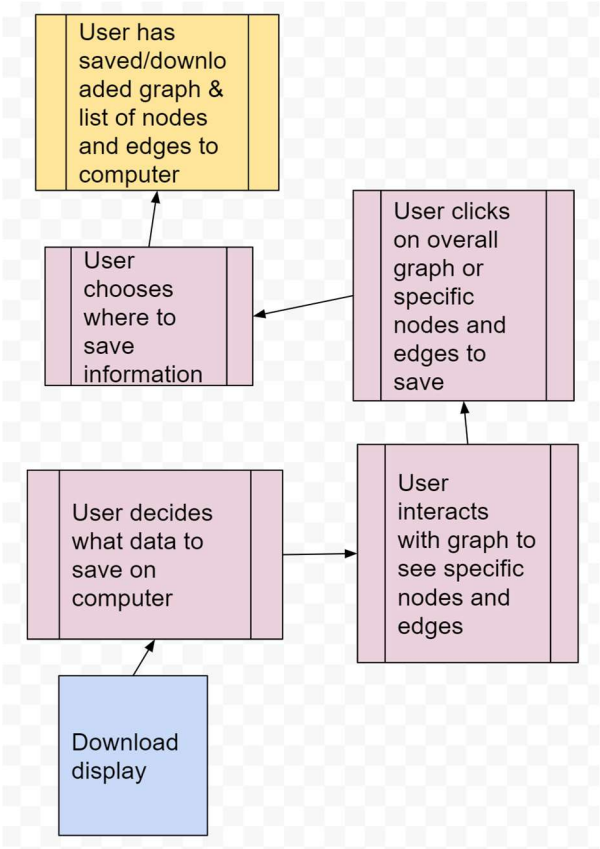

***Figure 9**: Flow chart showing how the "Download Display" service works*

- **Remove genomes -** While viewing a genome, and allow the user to:
    1. Remove it from their current genome database
    2. Remove its corresponding edges from the edge database.

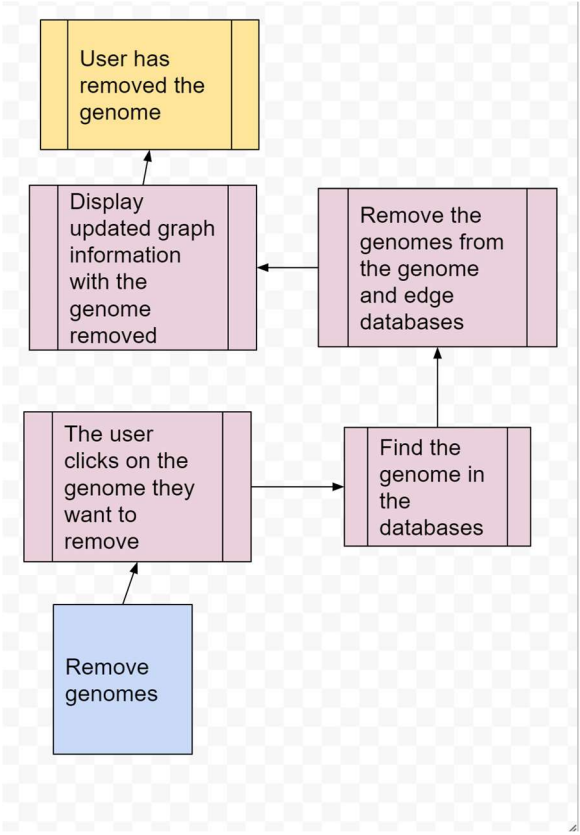

Figure 10: *Flow chart showing how the "Remove Genomes" service works*

In Table 1, we show the different input and output information and the different functionality each service provides. The service ID is there to differentiate between the different services.

Service ID	Service Name	Input information	Output information	Libraries; Functions; Environments	API endpoint (if applicable)
1	View Genomes	Genome CSV file	Genome data: name, number of edges, location on graph	NetworkX, Plotly	N/A
2	Search Genomes	Search category, list to search	Input list with only desired elements	SearchString, Range, Sort	N/A
3	Add Genomes	Genome CSV file	Genome list	Parse, Import, addToCSV	N/A
4	Compare Genomes	Genome CSV file	Data corresponding to both genomes	NetworkX, Plotly	N/A
5	Download Genomes/Edges	Genome/Edge list	Genome/Edge csv file	BioPython	N/A
6	Remove Genomes	Genome CSV file	Updated list with desired genomes removed	removeFromCSV	N/A

***Table 1**: Functions of each service provided by our application*

These goals show all the different operations each service uses and the output information.

Goal 1: View Genomes = Click and drag + Click on genomes + Show genome information

Goal 2: Search Genomes = Enter search term + Search database for desired genome + Act on genome (add, remove, etc.)

Goal 3: Add Genomes = Import .csv file with new genomes + Import data to current file of user + Sort genomes + Allocate edges

Goal 4: Compare Genomes = View multiple genomes + Select comparison method + Perform comparison

Goal 5: Download Display = View genomes on graph + Interact with graph to get desired image + Save graph and edges/nodes to computer

Goal 6: Remove Genomes = Remove a genome from the database + Remove edges from the database

User Interface

The user interface for this project was developed using the language Python and the packages NetworkX and Plotly. It also uses Dash and Pandas within Python. It takes in the data from two CSV files and analyzes them by column to form the graph. The first file node1.csv handles the information pertaining to each genome that appears when the node is hovered over on the graph. The second file edges1.csv handles the edges between nodes and additional information that is relevant to the graph such as the date. This file will also be used to help determine how to reduce the graph from all of the nodes to those within the specified date. This code handles the user interaction within the graph (i.e., zooming in or out, saving the graph, comparing genomes, and selecting parts of the graph). These are all handled with the plotly package within Python.

Mid-Application Data Manipulation

The Mid-Application Data Manipulation (dataManip.py) for this project was developed using the language Python using just the DateTime package. No package fits the very loose needs for the functions, so all of the functions written will need to be updated to keep their genericism, while trying to be as efficient as possible. To preserve the genericism of comparison in the datasets, any new functions written will also need to be written from scratch and many will need to be applicable to all possible data types.

Many of the methods written in the dataManip.py file contain a parameter called list, genomeList, or edgeList. The difference in naming convention is intentional, as certain methods are made for either the genomeList, or edgeList, or both. All of these parameters are going to be 2-dimensional lists (e.g., a 3x3 list is [[‘a’,’b’,’c’], [‘d’,’e’,’f’], [‘g’,’h’ ‘j’]]) in which every row is a genome, and every column is a different property of that genome. It is vital that the 2-dimensional lists be rectangular, where every row has the exact same number of elements as every other row. The same also applies for columns. This feeds into an even more common parameter, property. A property of a genome can be any element of that genome, whether it be useful or extraneous. Examples of properties include unique identifiers, geo-location, collection date, genome sequence, and host taxonomy ID. Some are more generally useful than others, but we need to be able to process them all. As a parameter, ‘property’, is an integer that determines the column of the genome collected, and thus the specific property to be used for the method.

Sorting and Search

The sorting functions are relatively simple and use the Mergesort algorithm to do the sorting. Aside from the 2-dimensional list and 'property' parameters, there is nothing unique as to how we are doing the sorting of any given list.

The search functions proceed linearly down the list. The generalized search functions are looking to see if the element contains the search term, so it does not have to be an exact match. When searching for a specific date, many date elements in the list may be incomplete. To solve this problem, the month values are assumed to be January and the day values are assumed to be the first of the month if they are unavailable.

Parsing

The parsing functions are probably the most complex of the functions in dataManip.py. Firstly, 'parserMultiFasta' is a parsing function that, when given a CSV file and configuration of the FASTA files, will parse through the CSV and FASTA files synchronously, taking the extraneous data from the csv and genome sequence from the FNA file and combining them into one complete genome [7]. The FASTA files do not represent the sequence as a pure string; the sequence is interspersed with newline ('\n') characters. So, we must cut these out and append the sequence together to obtain the proper genome sequence.

Range

For the range functions, as with the vast majority of the functions, they generally were written from scratch. Every one of the range functions is inclusive, so the 'earlier', 'later', and/or 'value' parameter values would be included in the result if they were to appear during the function processing. The first exception to this rule is 'removeGenomeEqual', where it wouldn't make sense to include the 'value' parameter, which you are specifically trying to remove. The second, and more complex exception is 'outsideRange'. We felt that since the whole point of this method is to exclude the elements that fall inside a range of values, and that is the polar opposite of 'insideRange', so therefore it should be exclusive.

Backend

As mentioned before the developer will need to have the package BioPython installed. BioPython is a very useful and helpful package in the development process as it has functions and methods to easily read a FASTA file, which is the common file type used when importing genome names and sequences. Functions also include parsing the FASTA file which is helpful in extracting specific genomes and adding them to wherever you would like to add them, which for this project's case is a CSV file. This functionality can be found in the "FASTA_input.py" file. Regarding the input file, comments specifically describe what each line of code does. "backend.py", is the main backend work file that helps with adding, removing, searching, and comparing genomes.

IX. Lessons Learned

Communication

One lesson we have learned is that clear communication with our client is key. It ensures that the end result is what the client wants. We've been able to communicate with our client through meetings, but more clear communication within the team could help the resulting application to be successful and useful to the client. Over communication we have found is not possible in a team project.

Project Design

We originally planned to use a traditional database backend to store and search the genomes. This direction proved to be not as efficient as we originally thought, as the frontend uses a CSV file to pull in the data needed for the visualization, which made the database backend not an ideal approach. It was simpler to move to a CSV oriented backend.

Our client, Dr. Heath, didn't mind us changing a major part of our project design. He agreed that this was a simpler solution than what we previously had, so we were able to make the change with little issue.

Schedule

In Table 2 we show the final timetable of our project. Each of the tasks shown were the milestones in our project. We show who completed each milestone and when the task was completed.

Tasks Accomplished	Completed By	Dates Completed
Design/planning of Front-end	Caroline Turner, Jared Hubert	February 8, 2021
Design/planning of Back-end	Jake Fisher, Drew Pompeii	February 10, 2021
Front-end working with 'dummy data'	Caroline Turner	February 15, 2021
Backend redesign to show change to CSV forward back-end	Jake Fisher, Drew Pompeii	February 28, 2021
Front-end and Back-end connection	Caroline Turner, Jared Hubert, Jake Fisher, Drew Pompeii	April 10, 2021
Front-end polished	Caroline Turner, Jared Hubert	April 10, 2021
Testing and bug fixes complete	Jake Fisher	April 29, 2021

Table 2: *Timetable of our project, who completed each milestone and when they completed them.*

X. Future Plans

Our plans include fixing the alignment software to check that the genome sequences are being properly analyzed. After we complete this, we have a few more plans for how this project could be continued and expanded. First, this project could be adapted to work with other viruses such as influenza or the common cold. It even could be adapted to study genomes of animals such as studying the mutations from purebred to inbred dogs.

Another way this project could be extended is through an extension of mutational analysis. As the SARS-CoV-2 virus develops, its mutations create different variants that have varying levels of immunity against its vaccines.

Finally, this project could continue by allowing users to personally select which data they want to be shown using check boxes and other form-like processes. This would mean that the user is not restricted by only getting the name of the genome, the location, collection date, nucleotide length, and genome length. They could have additional pieces of information to include or exclude based on what they would like to study. As shown in Figure 11, the user has imported a CSV file with the five columns: Genome Name, Isolate Name, Bioproject, US State, and Taxonomy Number. They have decided that the columns they would like to reduce the dataset by, however, are just Genome Name and Bioproject. And they only want to see genomes with the name: JQ3756f.1 and the Bioproject identifier: PD2710. This could be further expanded upon using symbols such as hyphens, commas, and quotation marks to allow the user to specify a range of values they would like to see (using a hyphen), a list of different values they would like to see (using commas), a specific value that contain another symbol (quotation marks), or a combination of those symbols. We believe that this would provide the user with much greater freedom in their choices and detail in their analysis. The only reason that this was not implemented is that we did not think of it early enough in the development process to incorporate it into our plans and did not have the time to add it towards the end of the project, but we do believe that this is the next logical step in the project and an idea from which more ideas may sprout.

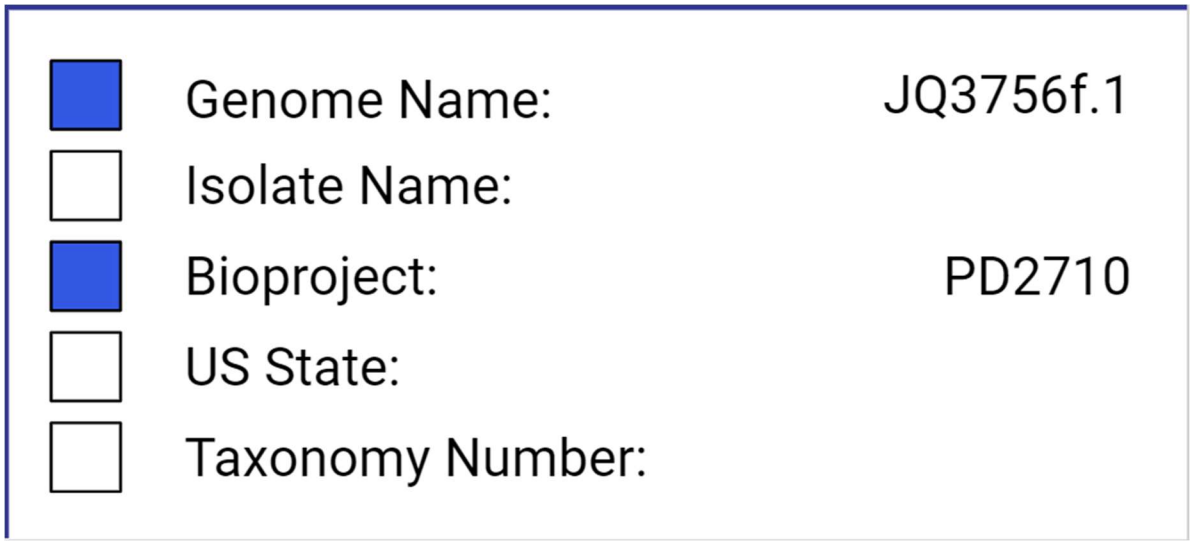

Figure 11: *A CSV file the user has uploaded and is choosing to reduce the columns down to "Genome Name" and "Bioproject"*

XI. Acknowledgements

We would like to thank our client, Dr. Lenwood Heath, and our professor Dr. Edward Fox for their continued help and support in the creation of this project. Their feedback was key to successfully creating the application.

Dr Lenwood Heath can be contacted at heath@vt.edu. Dr. Edward Fox can be contacted at fox@vt.edu.

XII. References

[1] National Human Genome Research Institute, *Genetics Glossary*, 2021. https://www.genome.gov/genetics-glossary/Mutation#:~:text=A%20mutation%20is%20a%20change,mutagens%2C%20or%20infection%20by%20viruses. Accessed February 10, 2021.

[2] J. Wang, *Python Interactive Network*, November 19, 2019. https://towardsdatascience.com/python-interactive-network-visualization-using-networkx-plotly-and-dash-e44749161ed7 Accessed February 15, 2021.

[3] Biopython, *Biopython*, 2021. https://biopython.org Accessed February 12, 2021.

[4] Y. Zhang, *FASTA Format*, 2021. https://zhanglab.dcmb.med.umich.edu/FASTA/ Accessed March 2, 2021.

[5] J. Chang et al., *Biopython Tutorial and Cookbook*, September 4, 2020. https://biopython.org/DIST/docs/tutorial/Tutorial.html Accessed February 25, 2021.

[6] Tutorials Point, *Biopython - Sequence Alignments - Tutorialspoint*, 2021. https://www.tutorialspoint.com/biopython/biopython_sequence_alignments.htm#:~:text=Sequence%20alignment%20is%20the%20process,region%20of%20similarity%20between%20them Accessed February 20, 2021.

[7] Biostar, *Correct Way To Parse A FASTA File In Python*, April 15, 2010. https://www.biostars.org/p/710/ Accessed March 20, 2021.