University of Alabama in Huntsville

Exploration of Genomic Predictions in Future Covid Variants Through AI (Part 1 of 2)

Undergrad IC CAE Semester Research

Michael Kelly

Computer Science Major, Biology Minor, Math Minor

Scholar of the IC CAE Consortium at Huntsville

Abstract

Since the declaration of the initial covid outbreak as a pandemic, the virus has yet to be eradicated by conventional medical means. While the development of the Pfizer, Moderna, and related vaccines in 2020 were a monumental step in curbing the virus, the rate of it's mutation has allowed it to avoid vaccine-induced herd immunity. As more time has passed, newer variants of the novel coronavirus have become more resistant to these keystone vaccines. The aim of this paper is to explore the possibility of using predictive AI technology to assist medical researchers in developing more potent vaccines against newer variants. In theory, utilizing such AI technology would allow biotechnology companies to predict the genetic make-up of potential future covid variants. In doing so, medical researchers could use this data to develop vaccines before a potential variant emerges in the real world (thus eliminating the time lag between outbreak and vaccine development).

The nature of this research follows closely to a recent study titled *Using artificial intelligence techniques for COVID-19 genome analysis* (Nawaz et al.). The goal of the study was to analyze six covid genome samples for recurring nucleotide sequences, pattern rules within each sequence, and the rate of mutation among the samples. The study concluded with insightful data on the genetic makeup covid-19; however, the information gathered did not support its methodology as being reliable for predicting potential covid nucleotide segments. This paper modifies several aspects of this study to develop a more reliable method for predicting covid genomic data. These major changes include increasing the sample size from 6 genomes to 23, differentiating each sample by their variant type (three samples representing a single variant,

with the exception of one), and comparing the similarities between variants in order of appearance in the real world.

Thus far, preliminary research for this project was conducted to identify the top variants of concern. Additionally, a Java-based program was developed to analyze the used covid genome samples for similar sequences in order of appearance in the real world. The dataset generated from this program was run through an open source sequential pattern mining software called *SPMF* to discover additional nucleotide similarities, frequent itemsets, and potential pattern rules. The results from these tests discovered several similar nucleotide fragments across the sample covid variants. This research has conducted two of its three intended phases, with the third phase projected to be completed by the end of Spring 2022. This phase will involve the training and development of an AI to predict a sequence of nucleotides using data gathered from the *SPMF* analysis. The final steps of this phase will involve the comparison of these generated nucleotide sequences against the genomes of more recent variants to determine the model's accuracy.

Introduction

Real World Problem

As we have seen with the coronavirus for the past two years, our healthcare system's ability to adapt to its viral variants has been suboptimal at best. As of August 12th 2021, the Delta variant is currently spreading throughout the US at a rapid rate (especially in the South). Current methods for combating this covid variant include the administration of vaccines initially made for the origin strain, masking in public spaces, and social distancing. Our healthcare system has been taking a reactive rather than a proactive approach in combating the covid virus. To more

efficiently curb the rates of infection, it would be beneficial if the genetic make-up of potential variants were accessed before they appear in the populus.

Solution

By utilizing cutting edge computer science technology (AI, Machine Learning, and/or Deep Learning), it may be possible to predict future variants of viruses before they arise in the real world. Ideally, these technologies would be used to assess multiple viral variants for patterns of recurring mutations in their genomes. These tests would develop a statistical value to represent the chance of mutation in each discovered problem gene. By finding these recurring mutations, it may then be possible to postulate which nucleotide sequences are likely to mutate in future variants. Using this data, medical researchers could develop vaccines for likely mutations ahead of time before major outbreaks occur. In doing so, resources spent on research and development during a pandemic can be reallocated to infection prevention, subsequently saving lives.

Variables to Consider

While this solution may appear simple on face value, there are several variables that must be considered for a project of this scope to be successful. One variable is finding the appropriate data to feed into a hypothetical computer model. Fortunately, there are several open-source genetic banks that can be used to acquire genomic information (ie the *National Center for Biotechnology Information*). The problem then becomes identifying what type of genetic information would be needed to create a statistically significant method for predicting the make-up of potential strains (i.e. assessing entire genomes of viral variants, analyzing select genes seen across all strains, common mutation sites, etc). Other factors to consider include the amount of genes affected in each variant and the rates of mutation for certain strains. There are

likely other hurdles not listed here that could further complicate a project of this scope. Ultimately, much additional research would be needed to fully understand the expanse of variables that could affect the development of a genomic prediction strategy.

Theory

While genetic mutations are random, some are better suited for survival than others in nature. The suitability of favorable traits can influence a virus to selectively keep favorable genetic alterations in future generations. Organisms pass on traits to future progeny based on the selective pressures of their environment. Similarly, viruses also pass on traits to future replicants based on favorable mutations that exploit their hosts. Viruses rely on random mutations to thrive in their environment much like organisms do, the major difference being that viruses are in a perpetual state of genetic change. The aim of this research is to identify the trend of nucleotide sequences seen in covid-19 variants to predict which fragments may be passed on in future strains through the usage of AI technology.

Phase 1: Preliminary Research

In this stage of the project, preliminary research was conducted to gather all of the information needed for the genome analysis of covid variants. Research began with finding scientific papers related to the objective of this project. While the topic of sequence prediction is relatively niche in the genomics field, there was one study in particular that aligned closely to the subject. The study, titled *Using artificial intelligence techniques for COVID-19 genome analysis,* explored different methods of analyzing the covid-19 genome using various sequential pattern mining techniques. The study was used as the framework for this research with several modifications made to take into account variants types and the order of which each appeared in the real world.

The study utilized an open source pattern mining software called *SPMF* to look for recurring sequences in covid genomes. This research follows some of the techniques used in the study's genome analysis process with minor alterations made to the software settings. The *World Health Organization's* website was used as a resource to determine the order of variant appearance among the covid samples being accessed. The *WHO's* website has documentation related to the first sightings of all known covid variants. To hone the scope of this project, only the first eight covid variants were chosen to be analyzed. These first eight variants of concern include Beta, Alpha, Delta, Kappa, Gamma, Iota, Eta, and Lambda. Using this information, a lineage of these covid variants was constructed to represent each one's emergence in the real world. After a lineage was established, the *National Center for Biotechnology Information's* public genome bank was used to acquire covid samples for each variant. Every variant in this research had three representative genome samples with the exception of Kappa, which only had two available at the time.

Phase 2: Software Development

The second phase of this research involved the development of a program that could identify common nucleotide sequences across the eight chosen covid variants. The software was written entirely in the Java programming language. The code logic can be broken down into three essential sub-phases: the Association phase, the Self-Similarity Identification phase, and the Parent-Child Similarity phase. The end result of this program's computations is a dataset with recorded nucleotide sequences that were commonly seen throughout the variant lineage. This dataset is duplicated in several formats to serve as inputs for the *SPMF* testing after program execution.

Association Phase

The Association phase of this program does not have any code logic within it. The goal of this phase is for the programmer to have an understanding of the order in which the covid variants will be analyzed. The first part of this phase is to identify which strain appeared first in the world and every proceeding variant after in sequential order. Variants emerging prior to the current strain in question are denoted as ***ancestors***, while variants appearing after are labeled as ***successors***. The bulk of this logic was already accomplished in phase 1 of this research. ***Figure 1.01*** below showcases the lineage constructed from variant documentation provided by the *World Health Organization*.

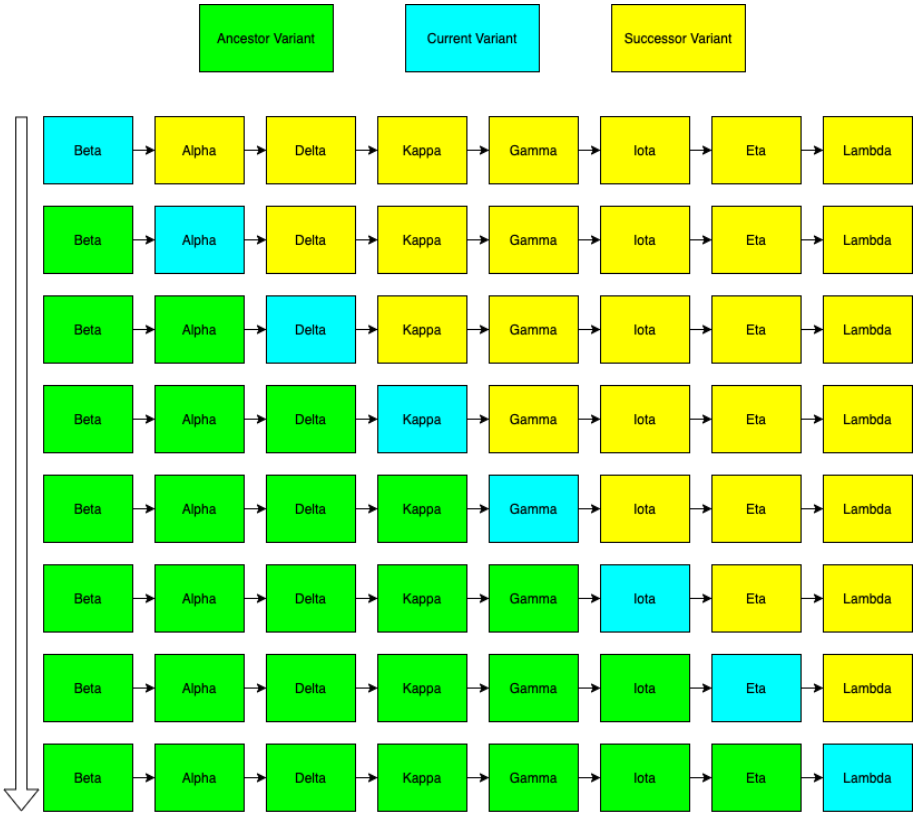

Figure 1.01

The second part of this sub phase is to establish parent-child relationships between each variant within the lineage. In a ***parent-child relationship***, the current strain in question is denoted as the ***parent***, while its immediate successor is denoted as the ***child***. In the case of the lineage displayed in ***Figure 1.01***, the seven (Parent, Child) relationships are: (Beta, Alpha) ; (Alpha, Delta) ; (Delta, Kappa) ; (Kappa, Gamma) ; (Gamma, Iota) ; (Iota, Eta), and (Eta, Lambda).

Self-Similarity Phase

The goal of the self-similarity phase is to establish an average representative genome for each variant being analyzed. This is necessary because viruses of the same variant type may still contain slight genetic differences. This representative average of a variant's genome is defined as a ***strain average***. The strain average is calculated by comparing three of the samples associated with each variant against one another and reconstructing a modified genome containing commonly found sequences. The strain averages of each variant are then compared against each other in lineage order in the next phase of the progam.

There were several notable challenges that arose during the implementation of the strain average computational logic. The first dilemma was devising an efficient algorithm to find every common substring among the sample genome strings. Interestingly, each of the three genome samples representing every covid variant began with the same nucleotide sequences. This unique attribute greatly helped in developing an appropriate algorithm for the comparisons. The next problem was that each variant sample in this project varied in base pair length and began in different positions on the genome. Another interesting characteristic used to solve the size obstacle was that all of the covid-19 genome samples were less than 30,000 nucleotide base pairs in length. To address the varying sizes, the two longest samples of the three representing each

variant had their extrateraneous starting sequences spliced to match the starting position of the shortest genome. To ensure that the ending sequences were also aligned, placeholder space values were concatenated to the end of the shorter strain samples. The last challenge that needed to be surmounted was handling the nucleotide misreads that the genomic data from the *National Center for Biotechnology* occasionally contained. The nucleotide base pairs that normally construct a genome are 'A', 'T', 'G', and 'C'. The *NCBI* represented these nucleotide misreads as 'n' within the affected genomes. To ensure that these misreads did not affect the comparison process, code logic was implemented to leave them out of the averaging. ***Figure 1.02*** shows a high-level overview of the algorithm employed to calculate the strain average.

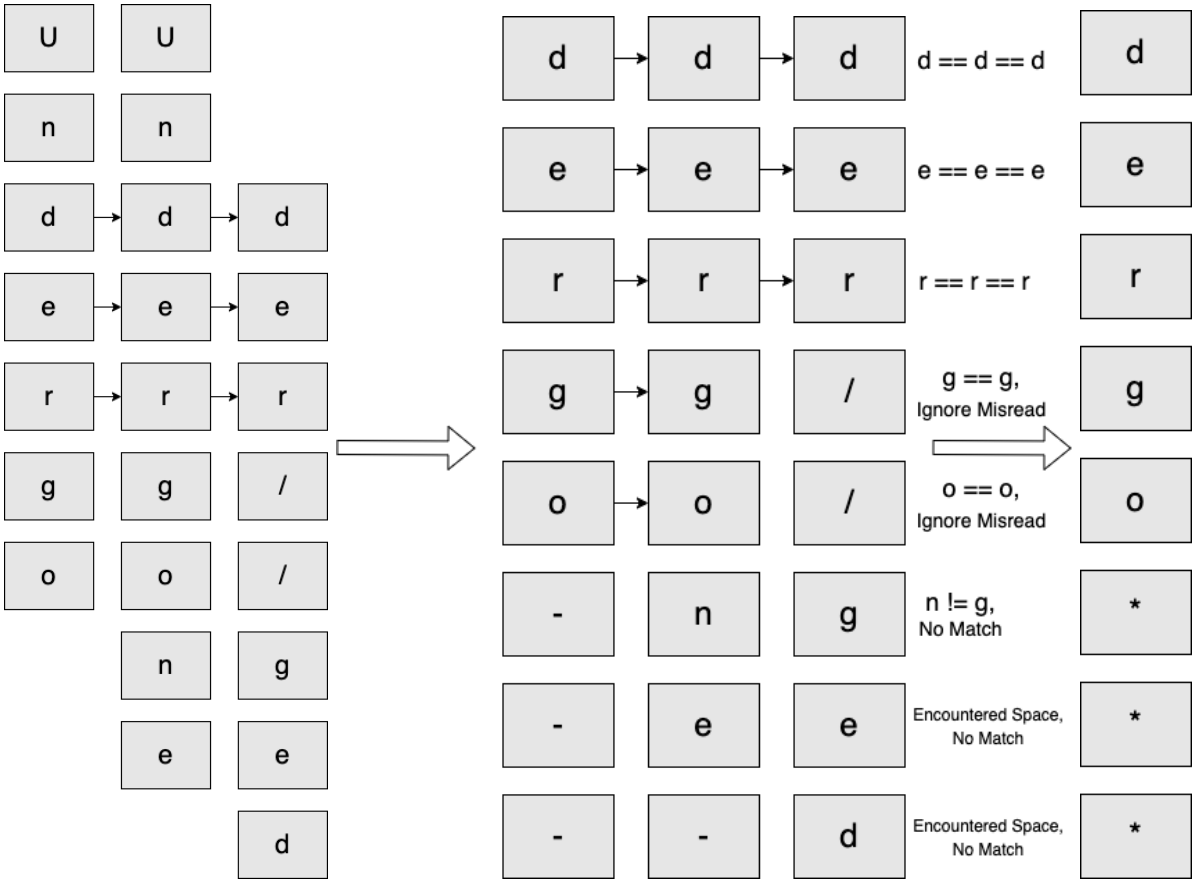

Figure 1.02

Rather than comparing three 30,000 nucleotide base pair genomes, ***Figure 1.02*** uses the strings “Undergo”, “Undergone”, and “deranged” to simplify the inputs. Additionally, the string “deranged” contains two misreads on its ‘a’ and ‘n’ characters, which are denoted as ‘/’. The first part of the comparison involves aligning the strings together and determining a common starting sequence (in this case, “der”). The extraneous “Un” found in strings “Undergo” and “Undergone” are spliced out to form “dergo” and “dergone” respectively. Next, the strings are further modified to match in size. In this example, placeholder space values denoted as ‘-’ are appended to “dergo” and “dergone”. The current state of the strings are: “dergo---”, “dergone-”, and “der//ged”. It is important to note that all of the strings now start on the same sequence and are of the same length. The final portion of this algorithm compares each character across the three strings. If all characters across each string are the same in one position, such as the ‘d’ character in the first position, that character will be copied over to a new string. If a misread is being compared to a valid character, such as the two ‘g’ characters in “dergo---” and “dergone-” to the misread in “der//ged” in the fourth position, the misread is left out of the comparison process and only the two valid characters are compared. Since ‘g’ and ‘g’ are the same in this case, the character is concatenated to the new string. If a placeholder space is being compared to a valid character, such as the two ‘e’ characters in “dergone-” and “der//ged” to the ‘-’ character in “dergo---” in position 7, the comparison yields the same result as two mismatching characters being read. In the case of two mismatching characters, such as ‘a’ compared to ‘b’, a mismatch placeholder denoted as ‘*’ is appended to the new string. The ending strain result of this string comparison is “dergo***”.

Parent-Child Similarity Phase

The Parent-Child Similarity phase portion of this program is where the majority of computations take place. The goal of this phase is to find similar sequences between each parent-child's strain average in lineage order. The end result of this phase is a dataset containing all frequently occurring sequences in the lineage. This dataset is generated in multiple formats to ensure compatibility with the *SPMF* pattern tests conducted after program execution. The main challenge with this phase was implementing an algorithm to find all common substrings between two strain averages. Due to the large amount of genomic differences between variants, the algorithm used to find strain averages could not be employed here. The strain averages between covid variants differed in starting positions and overall size, preventing the use of certain properties from the former algorithm to manipulate string comparisons. Early in the software's development, a brute force approach was implemented to analyze all common substrings within each strain average. Due to the inefficiency of this algorithm, computation time ranged between 3-4 hours. ***Figure 1.03*** showcases this approach from a high-level perspective.

Example Strings: [“longer”, “german”]

Nested Loop 1: Finding All Substrings in “longer”

	Inner Loop 1	Inner Loop 2	Inner Loop 3	Inner Loop 4	Inner Loop 5	Inner Loop 6
Outer Loop 1	l	lo	lon	long	longe	longer
Outer Loop 2	o	on	ong	onge	onger	
Outer Loop 3	n	ng	nge	nger		
Outer Loop 4	g	ge	ger			
Outer Loop 5	e	er				
Outer Loop 6	r					

Nested Loop 2: Finding All Substrings in “german”

	Inner Loop 1	Inner Loop 2	Inner Loop 3	Inner Loop 4	Inner Loop 5	Inner Loop 6
Outer Loop 1	g	ge	ger	germ	germa	german
Outer Loop 2	e	er	erm	erma	erman	
Outer Loop 3	r	rm	rma	rman		
Outer Loop 4	m	ma	man			
Outer Loop 5	a	an				
Outer Loop 6	n					

Nested Loop 3: Comparing Substrings from "longer" to "german"

```
for substringA in "longer"
    for substringB in "german"
        if substringA = substringB
            save commonSubtring
```

Similarities: g, e, r, n, ge, er, ger

Nested Loop 4: Removing Duplicates From Common Substrings (Part 1)

```
for substringA in commonSubstrings
    for substringB in commonSubstrings
        if (substringA is not substringB) and (substringB contains
        substringA)
            remove substringA from commonSubtrings
            (NOTE: removed from a copy of the common list)
```

Nested Loop 5: Removing Duplicates From Common Substrings (Part 2)

```
for substringB in commonSubstrings
    for substringA in commonSubstrings
        if (substringB is not substringA) and (substringA contains
        substringB)
            remove substringB from commonSubtrings
            (NOTE: removed from a copy of the common list)
```

Filtered Similarities: ["ger", "n"]

Figure 1.03

As seen in the figure above, this brute force method finds all common substrings at the great cost of performance efficiency. This may work for smaller inputs such as strings “longer” and “german”, but when dealing with genomes nearly 30,000 nucleoide base pairs long, the time per comparison grows exponentially. Due to this reason, a secondary algorithm was devised to improve computation efficiency. ***Figure 1.04*** highlights this newer approach.

Example Strings: [“longer”, “german”]

Strings		l	o	n	g	e	r
	0	0	0	0	0	0	0
g	0	0	0	0	1	0	0
e	0	0	0	0	0	2	0
r	0	0	0	0	0	0	3
m	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0
n	0	0	0	1	0	0	0

Found Similarities: [“ger”, “n”]

Red = *Case for Empty String*
Green = *Found Substring*

Figure 1.04

This secondary approach employs dynamic programming to find all common substrings between two input strings. For this method to work properly, both inputs must be of the same length. To ensure this was the case when comparing two strain averages, the shortest of the two had placeholder space values appended to it to match the length of the longest input. Once the sizes are adjusted, one input is aligned on the x-axis of a 2d matrix while the other is aligned on the

y-axis. At this point, the code will check every entry of this matrix to determine if a character on the x-axis aligns with an equal character on the y-axis. For cases where two characters are not the same on either axis, the matrix entry is filled with a 0. If two characters are a match, then the entry is filled with a 1. If another matching character is found in the next lower diagonal entry, this entry will have the previous upper diagonal value incremented by one stored inside it. This is done to indicate length of the common substrings. Once all entries are filled, every diagonal of the matrix is then accessed for any non-zero sequences. Every non-zero diagonal represents a common substring. This algorithm quartered the computation time from 4 hours to approximately 1 hour.

After all parent-child groups completed their assessments, all commonly found sequences across the lineage were compiled into one dataset called the lineage average. The lineage average was then exported in several formats to external files for further *SPMF* testing. The file labeled "lineage-average.txt" showcases the lineage average in nucleotide format, where 'A', 'T', 'C', and 'G' represent the standard base pairs and '@' denotes the ending/starting position of two sequences. The text file labeled "CSPAM-TKS-ERMINER-input.txt" contains the same dataset, with '1' representing 'A', '2' representing 'T', '3' representing 'G', '4' representing 'C', and '-1' indicating a space between two nucleotides. This file is used as the input file for the CM-SPAM, TKS, and ERMiner tests in the *SPMF* software. The file labeled "Apriori-input.txt" follows the same format as "CSPAM-TKS-ERMINER-input.txt" with the absence of '-1' to indicate a space between nucleotides. This file is used as the input file for the Apriori test that is also conducted within *SPMF*.

SPMF Testing

The final half of phase 2 involves running the lineage average inputs into the *SPMF* pattern mining software for further sequence analysis. *SPMF* contains several algorithms that specialize in different forms of pattern mining. The three categories of focus for this research were sequential pattern mining, sequential rule mining, and itemset mining. Sequential pattern mining is primarily centered around finding commonly occurring sequences in a large data set. Itemset mining finds subsequences that make up these commonly found sequences. Sequential rule mining identifies frequent combinations of itemsets used to construct the larger common sequences.

Sequential Pattern Mining

The two algorithms of choice used for the sequential pattern testing were CM-SPAM and TKS. CM-SPAM identifies every possible combination of sub-sequences in a data set based on a set length. Due to the hardware limitations of my windows workstation (consisting of 16 gigabytes of RAM), the maximum size of sequences that could be computed on the CS-SPAM test were 9 nucleotides in length. This test analyzed all 262,144 possible subsequences at this set size. ***Figure 1.05*** below showcases the first and last 5 results of the analyzed nucleotide sequences. The results indicate that every subsequence tested appeared almost equally frequent as each other. This data is concerning, as it might suggest that the size limit of each fragment was not set large enough to capture the longer similar sequences being sought after. Rather than all sequences having a uniform rate of occurrence, it was expected for certain sequences to have more appearances than others.

Raw Output	Translated Output	# Of Times Seen
1 -1 1 -1 1 -1 1 -1 1 -1 1 -1 1 -1 1 -1 1 -1	A A A A A A A A A	338
1 -1 1 -1 1 -1 1 -1 1 -1 1 -1 1 -1 1 -1 2 -1	A A A A A A A A T	321
1 -1 1 -1 1 -1 1 -1 1 -1 1 -1 1 -1 1 -1 3 -1	A A A A A A A A G	326
1 -1 1 -1 1 -1 1 -1 1 -1 1 -1 1 -1 1 -1 4 -1	A A A A A A A A C	329
1 -1 1 -1 1 -1 1 -1 1 -1 1 -1 1 -1 2 -1 1 -1	A A A A A A A T A	326
...	...	...
4 -1 4 -1 4 -1 4 -1 4 -1 4 -1 4 -1 3 -1 4 -1	C C C C C C C G C	301
4 -1 4 -1 4 -1 4 -1 4 -1 4 -1 4 -1 4 -1 1 -1	C C C C C C C C A	300
4 -1 4 -1 4 -1 4 -1 4 -1 4 -1 4 -1 4 -1 2 -1	C C C C C C C C T	299
4 -1 4 -1 4 -1 4 -1 4 -1 4 -1 4 -1 4 -1 3 -1	C C C C C C C C G	297
4 -1 4 -1 4 -1 4 -1 4 -1 4 -1 4 -1 4 -1 4 -1	C C C C C C C C C	299

Figure 1.05

To analyze larger common sequences, the TKS algorithm was employed. This algorithm works functionally as CS-SPAM does; however, it allows the user to set a limit for the amount of results generated. To analyze sequences of 12 base pairs in length, a limit of 500 results had to be set for my workstation to fully compute the data. ***Figure 1.06*** highlights the first and last 5 results of the test. Once more, the results indicate that every subsequence appeared almost equally frequent to each other.

Raw Output	Translated Output	# Of Times Seen
1 -1 2 -1 4 -1 3 -1 1 -1 1 -1 1 -1 2 -1 1 -1 4 -1 2 -1 2 -1	A T C G A A A T A C T T	310
4 -1 2 -1 4 -1 2 -1 2 -1 1 -1 1 -1 1 -1 2 -1 4 -1 2 -1 2 -1	C T C T T A A A T C T T	310
1 -1 1 -1 1 -1 1 -1 1 -1 4 -1 2 -1 2 -1 2 -1 2 -1 1 -1 3 -1	A A A A A C T T T T A G	310
4 -1 2 -1 1 -1 1 -1 2 -1 2 -1 1 -1 2 -1 2 -1 2 -1 2 -1 2 -1	C T A A T T A T T T T T	310
2 -1 1 -1 3 -1 1 -1 4 -1 1 -1 1 -1 2 -1 1 -1 1 -1 2 -1 2 -1	T A G A C A A T A A T T	310
...	...	...
1 -1 2 -1 1 -1 2 -1 1 -1 2 -1 1 -1 1 -1 1 -1 1 -1 1 -1 3 -1	A T A T A T A A A A A C	310
1 -1 2 -1 1 -1 1 -1 4 -1 1 -1 1 -1 2 -1 1 -1 2 -1 2 -1 2 -1	A T A A C A A T A T T T	312
1 -1 1 -1 1 -1 3 -1 2 -1 2 -1 3 -1 1 -1 2 -1 2 -1 2 -1 2 -1	A A A G T T G A T T T T	310
1 -1 1 -1 3 -1 2 -1 3 -1 4 -1 2 -1 2 -1 2 -1 2 -1 3 -1 2 -1	A A G T G C T T T T G T	312
1 -1 1 -1 2 -1 2 -1 2 -1 2 -1 1 -1 2 -1 1 -1 1 -1 3 -1 3 -1	A A T T T T A T A A G G	311

Figure 1.06

Item Set Mining

For the itemset mining portion of the *SPMF* testing, the ERMiner algorithm was utilized. This test found 15 potential itemsets that were used to create the previously found common sequences. ***Figure 1.07*** showcases all of the identified itemsets. Unsurprisingly, the individual nucleotides have the most frequent occurrences, as they make up all possible sequences in the genome. It is also predictable that larger itemsets, such as ones containing three or more nucleotides, would appear less often than smaller item sets. An interesting observation to make note of is the difference in occurrences between certain nucleotide pairs, such as ‘AT’ and ‘GC’. In this case, itemset ‘GC’ appears four times more frequently than ‘AT’ does.

Raw Output	Translated Itemset	# of Times Seen
1	A	37,636
2	T	40,345
3	G	24,635
4	C	22,568
1 2	A T	81
1 3	A G	136
1 4	A C	242
2 3	T G	195
2 4	T C	323
3 4	G C	354
1 2 3	A T G	43
1 2 4	A T C	81
1 3 4	A G C	128
2 3 4	T G C	165
1 2 3 4	A T G C	43

Figure 1.07

Sequential Pattern Rule Mining

The last test conducted in this phase involved the analysis of frequent itemset combinations used to build commonly found sequences. The Apriori algorithm was used in this regard. ***Figure 1.08*** showcases 9 of the 50 total results compiled. Similar to the ERMiner algorithm, it is expected for results involving smaller itemsets to be statistically more frequent than their larger counterparts. Interestingly, the accuracy values of all 50 results never faltered below 74.8%, indicating that these itemset rules are fairly consistent.

Raw Output	Left Itemset	Followed By	Right Itemset	# Times Seen	%Accuracy
1 ==> 2	A	->	T	773	96.5%
2 ==> 3	T	->	G	728	90.8%
3 ==> 4	G	->	C	638	81.6%
1,2 ==> 3	A T	->	G	716	89.5%
2,3 ==> 4	T G	->	C	621	79.4%
1,3,4 ==> 2	A G C	->	T	650	90.5%
2,3,4 ==> 1	T G C	->	A	622	86.4%
4 ==> 1,2,3	C	->	A T G	602	81.2%
3 ==> 1,2,4	G	->	A T C	598	76.5%

Figure 1.08

Phase 3: AI Development

Currently, this research has completed two of its three initial phases. The goal of phase 3 is to develop an AI using the *SPMF* data to predict what a potential covid variant's genome could look like. The possibility of using recurrent neural networks for the prediction process is currently being explored with additional preliminary research. Likewise, the implementation of image recognition AI in phase 2 as a part of the lineage analysis process is also being explored. As of now, an experimental build of the program used in phase 2 has the ability to convert similar nucleotide sequences between the covid Beta and Alpha variants into a color coded image file. ***Figure 1.09*** showcases the resultant image produced from this parent-child comparison. In the figure below, each colored stripe represents a specific nucleotide: red matching to A, green to T, blue to G, and Yellow to C.

Figure 1.9

In addition to the AI development in phase 3 of this research, it may be necessary to revise some of the steps in previous phases and generate updated results. For example, the tests conducted in the *SPMF* portion of phase 2 could benefit from a computer build with larger RAM capacity and better CPU capabilities. With these upgrades, longer sequences of similar nucleotides patterns could be identified from the lineage average. This would in turn enhance the accuracy of the recurrent neural network AI that will be used for the genome prediction process. Likewise, modifications to the preliminary research in phase 1 may also be necessary. As more information on covid-19 comes out, a better understanding of how each variant relates to one another and their appearances in the world could affect the interpretation of the gathered results. Including additional variants in this research is another avenue that could potentially be explored as covid continues to mutate. A final consideration that may be explored in phase 3 or beyond is the

addition of another dataset to be produced in phase 2. Currently, the singular dataset from this phase outputs similar nucleotide sequences found across a covid variant lineage. Perhaps a secondary dataset could be generated to find the differences across the lineage and further examined for similarities via *SPMF*.

Conclusion

While still far away from a reliable method of predicting a covid variant's genome, these first few steps into the matter helped reveal several key areas that need to be explored for such a possibility. This research has currently completed two of its intended three phases. In phase 1, preliminary research on the first eight covid-19 variants in the world was conducted. Phase 2 compiled this information into a structured format and was analyzed through a variety of sequence tests. Results from these tests revealed that all possible nucleotide combinations of length 9 to 12 base pairs in length appeared uniformly across the variant lineage. This phenomenon is likely due to the sequence limit being set too low to capture the longer sequences being sought after. This can be fixed by reconducting these tests on a computer with higher end hardware specifications to raise the sequence length limitation. The produced results also identified itemsets that made up the commonly found sequences and the pattern rules associated between them. Moving forward, several aspects of the first and second phases will be revised to produce more accurate results from the *SPMF* testing. Once these refinements are in place, phase 3 will move into the development of a recurrent neural network AI to predict the potential genomes of future covid-19 variants. This AI model will be trained using the pattern rule data produced from the *SPMF* testing conducted in phase 2.

Citations

Kelly, M. (n.d.). Covid-Genome-Analyzer-Prototype: Proof of concept prototyping related to my genomic research at UAH. (work in progess). GitHub. Retrieved November 29, 2021, from https://github.com/MPKellyy/Covid-Genome-Analyzer-Prototype.

Nawaz, M.S., Fournier-Viger, P., Shojaee, A. *et al.* Using artificial intelligence techniques for COVID-19 genome analysis. *Appl Intell* 51, 3086–3103 (2021). https://doi.org/10.1007/s10489-021-02193-w.

U.S. National Library of Medicine. (n.d.). *GenBank Overview*. National Center for Biotechnology Information. Retrieved November 28, 2021, from https://www.ncbi.nlm.nih.gov/genbank/.

World Health Organization. (n.d.). *Tracking sars-COV-2 variants*. World Health Organization. Retrieved November 28, 2021, from https://www.who.int/en/activities/tracking-SARS-CoV-2-variants/.

SPMF: A java open-source data mining library. (n.d.). Retrieved November 28, 2021, from https://www.philippe-fournier-viger.com/spmf/.

Fournier-Viger, P., Gomariz, A., Campos, M., & Thomas, R. (2014). Fast vertical mining of sequential patterns using co-occurrence information. *Advances in Knowledge Discovery and Data Mining*, 40–52. https://doi.org/10.1007/978-3-319-06608-0_4.

Fournier-Viger, P., Gomariz, A., Gueniche, T., Mwamikazi, E., & Thomas, R. (2013). TKS: Efficient mining of top-K sequential patterns. *Advanced Data Mining and Applications*, 109–120. https://doi.org/10.1007/978-3-642-53914-5_10.

Fournier-Viger, P., Gueniche, T., Zida, S., & Tseng, V. S. (2014). Erminer: Sequential rule mining using equivalence classes. *Advances in Intelligent Data Analysis XIII*, 108–119. https://doi.org/10.1007/978-3-319-12571-8_10.

Agrawal R, Srikant R (1994) Fast algorithms for mining association rules in large databases. In: Proceedings of Very Large Databases (VLDB), pp. 487-499