Stochastic Optimization using the Genetic Algorithm

Simon Austin Lee* Marcella Gomez Vanessa Jönnson
Jack Baskin School of Engineering
University of California, Santa Cruz
siaulee@ucsc.edu, mgomez26@ucsc.edu, vjonsson@ucsc.edu

Abstract

Stochastic optimization methods refer to the use and generation of random variables to compute solutions for complex combinatorial optimization problems. In computer science, there remains a major unsolved hypothesis called the *"P versus NP problem"*, which discloses whether every problem whose solutions can be quickly verified, can also be quickly solved for. *P* stands for polynomial time, which roughly means a set of relatively easy problems to be solved for. *NP* stands for non-deterministic polynomial time which is a set of difficult problems to be solved for. So, if $P = NP$, this would imply that the difficult set of problems have relatively easily computed solutions. So we apply this onto a *NP-hard* bioinformatics problem called the *Phylogenetic Tree reconstruction*, where we wish to explain the relations of the SARS-CoV-2 Deltacoronavirus epidemic directly from 10 different DNA genomes *(courtesy of the GenBank database owned by National Center for Biotechnology Information (NCBI) of USA, www.ncbi.nlm.nih.gov)*. The issue that arises from having 10 genome samples is that, there are 34,459,425 binary rooted trees to choose from. Therefore we use a metaheuristic called the *Genetic Algorithm*, which specializes in navigating massive search spaces to find the most optimal solution. This genetic algorithm is combined with another popular bioinformatics tool called the *Multiple Sequence Alignment* which is morphed into a method called *SAGA* or Sequence Alignment by Genetic Algorithm. By using this method we are able to compute a cluster-based distance matrix and feed it to the *Neighbor-joining* algorithm, where we can find and produce an optimized tree to solve a NP-hard problem in O(log n) time complexities which are extremely efficient and computationally quick for a problem that would take a lifetime if brute-forced. By following these procedure we were able to form our phylogenetic tree. This research is in our effort to understand the SARS-CoV-2 Deltacoronavirus and see the evolutionary tracks and similarities between the genetic sequences of different species and identify any correlations or outgroups between sequences.

1 Introduction

Stochastic Systems have wide applications in biology, physics, computer science and engineering especially in that of *combinatronics*, which is the area of mathematical studies that focuses on counting to obtain the results and specific properties of finite structures. What makes a system *stochastic* is its subject to being effected by noise which will be discussed in Section 2.4. So in simpler terms, finding the best combination of *things*, which will be a term used loosely to describe processes from a birds-eye view, can lead to optimized solutions for problems which simply cannot be solved for in any other way.

A clear cut example of such optimization could be seen through the *Knapsack problem*, which given a set of items, their corresponding weights, and a value, the goal is to determine the number of items to include in a collection so that the total weight remains less than or equal to the given limit while still trying to take a value as large as possible. The issues that arises within the *Knapsack Problem*, lies when there is some large *n* integer of items to be chosen from, in which trying to compute all the combinations by hand loses efficiency. This whole idea of efficiency leads to one of the largest unsolved theoretical hypotheses in computer science called the *P equals NP problem* [1], where it asks whether every problem whose solution can be quickly checked for correctness can also be computed quickly. An answer to the *P versus NP* question would open up whether problems that can be verified in polynomial time (*denoted as P*) can also be solved in polynomial time. If it turns out that P ≠ NP, which is the more popular belief, it would mean that there are problems in non-deterministic polynomial time (*denoted*

*Supported in part by the UCSC Genomics Institute as well as the Directed Reading Program of the UCSC Mathematics Department

as NP) that are harder to solve than to verify: they could not be solved in polynomial time, but the answer could be verified in polynomial time.

Therefore, a solution to the *Knapsack problem* and other time-consuming problems, is the focus of this paper using a method called the *genetic algorithm*, which is widely used in stochastic optimization because it is great at taking large, potentially massive search spaces and navigating through them, finding the solutions, to problems you might not find otherwise in a lifetime. It takes the possibility of approaching these *NP-hard* problems, and finds a very accurate approximations for them.

2 The Genetic Algorithm

The Genetic Algorithm (GA), is a metaheuristic algorithm inspired by *Charles Darwin's* theory of *Natural Selection*, used for solving optimization and search problems in machine learning. This algorithm's strict purpose is to solve for problems that would take a long time to solve, often producing one of the best approximated solutions. The ability to begin solving *NP-hard* problems is just scratching the surface and with this algorithm there is an expectation for more complex systems and hypotheses to be approached in this manner. *The following example in Sections 2.1-2.4 is the GA applied to a very basic simulation of the Knapsack Problem. Code is written all in Python for simplicity and easily readable code.*

2.1 The Setup

The way the genetic algorithm works is through bits (or nucleotide/amino bases) represented in binary showing membership to a set, where 0 is denoted as not being a member of a particular set, and 1 being a member of a particular set. The use of binary is in efforts to reach a goal to emulate a genetic representation of a solutions domain. Before populating a solutions domain, three factors must be determined: how long each bit sequence or *genome* is to represent the n amount of individual *things* or *phenotypes* that are to be selected from, how many possible solutions we want to be randomly produced, and a limit that if succeeded will disqualify it as a valid solution. For example, the singular binary sequence *1001010111* which signifies a single possible solution, shows a bit length of 10 resembling 10 possible phenotypes within a genome, while we see that 6 phenotypes are members of this solution, simply by counting the number of 1's that are present in the sequence. It is important to note that the genome length never increases or decreases because there are a fixed amount of phenotypes present, and that any number of bits can be flipped to a 1 as long as it remains less than or equal to the limit set. Once a limit and integer n are chosen, n amount of unique solutions will be produced of some selected arbritary length l. This population of random possible solutions are what we refer to as a *generation* and this first generation is called generation 0 (generations can be tracked by *i-1 generations*, where i marks the number of iterations). Furthermore, because all the possible solutions were randomly generated, this evolutionary process begins in complete *chaos*.

Next begins an iterative procedure that will continue to reproduce as long as there are more than one non-zero solutions from the pervious generation. A solution is changed to zero if and only if the total sum of the collection of items succeeds the limit. So in order to reproduce even more different solutions, the *Natural Selection* process is introduced, where *fitness, single-point crossovers, and mutations*, all concepts from *Darwin's* Theory of Evolution are assessed.

2.2 Fitness

```
def fitness(genome: Genome, things: List[Thing], weight_limit: int) -> int:
    '''
    This evaluates a solution and sees its fitness level and whether it succeeds the
        limit that was established
    '''

    weight, value = 0

    for i, thing in enumerate(things):
        if genome[i] == 1:
```

```
            weight += thing.weight
            value += thing.value

            if weight > weight_limit:
                return 0

    return value
```

To begin, the fitness function's primary purpose is to assess a fitness value (the closer it is to the limit, the better), which essentially tells the person how good of an approximation a given solution actually is. It begins by first checking that the string length of the genome, is equivalent to that of the number of items (phenotypes) present. Next we initializes a weight, and value to 0 so the sum can always be calculated correctly. Finally as it iterates through the for loop, the function will check the bits that have been flipped to 1, and add those weights and value of items up, checking whether it succeeds the limit. The return value of this function is the list of items as long as it complies with the weight limit, where if it succeeds it simply just returns 0.

2.3 Single-point Crossovers

```
def single_point_crossover(a: Genome, b: Genome) -> Tuple[Genome,Genome]:
    '''
    performs the exchange of n length bits to be swapped to produce new solutions
    '''
    if len(a) != len(b):
        raise ValueError("genome and things must be of same length")

    length = len(a)
    if length < 2:
        return a, b

    p = randint(1, length -1)
    return a[0:p] + b[p:], b[0:p] + a[p:]
```

After evaluating a fitness score, the program will look to perform the single point crossover. In evolutionary biology, gene conversion can be *allelic*, meaning that one allele of the same gene replaces another allele from another gene. Similarly in this Python function, we perform a swap of a random number of bits in the range $1 \leq x \leq$ (length of genome - 1). This will then swap a portion of the two randomly selected parent genomes with one another usually resulting with a better approximation that is closer to the limit. These two newly generated solutions begin what would be the next generation (Generation 1 in our case). We repeat this process as long as there are non-zero solutions from the previous generation to complete this new generation.

Additionally, whats so fascinating about this algorithm is that in both nature and computers, it simulates and generates better offspring/solutions from generation to generation which models the *Natural Selection* behavior almost perfectly. However, this algorithm unfortunately is not perfect because our single-point crossover and selection functions are governed by randomness. The reason randomness is an issue is because there is no way to guarantee that we won't destroy our best solutions produced from each preceding generation. To resolve this issue, we must introduce a process called *elitism* [2]. Elitism can be described as, the most fit (having a fitness score close to the limit) handful of solutions are guaranteed a place in the next generation - without undergoing mutation. So, in order to preserve the most fit solutions, we must select another self selected number of n-top solutions, whose genomes will be copied into is successive generation.

2.4 Mutation

```
def mutation(genome: Genome, num: int = 1, probability: float = 0.5) -> Genome:
```

```
    '''
    randomly flips a bit within a genome to behave like a mutation
    '''
    for _ in range(num):
        index = randrange(len(Genome))
        genome[index] = genome[index] if random() > probability else abs(genome[index] -
            1)
    return genome
```

At last, we have *mutations*. Mutations in evolutionary biology are simply a change in a sequence of an organism's DNA. However, since we are not working with nucleotide bases in this example but rather with bits, we must also familiarize ourselves with *noise*, a concept that dates back to an interests of Einstein (1905). In the communication domain, noise (unwanted random disturbances) make it difficult to have a trivial signal. These fluctuations occur when there is a suspected origin that implicates the action of a very large number of "degrees of freedom" or variables. The coupling of noise to nonlinear systems can lead to non-trivial effects like, unstable equilibria and shift bifuracations [3]. Therefore it is widely believed that noise drastically modifies the deterministic dynamics of this system adopting its stochastic qualities. So in our case, noise behaves similarly to mutations where a singular random bit might get flipped. The way we simulate this behavior on a computer is by random probability. In the Python function above, we run a single iteration of the for loop, with a probability of your choice, in our case 0.5 to determine whether a random bit gets flipped. Though it may be possible to destroy one of our best solutions, mutations are essential to evolution. The new genetic variant (allele) spreads via differential reproduction and is a defining aspect of evolution. So although there is some random probability to destroy our best fit genome, there is also an equal chance that we construct an even better solution.

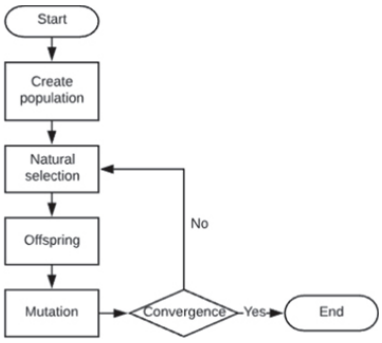

Figure 1: A flowchart of the Genetic Algorithm [4]

After the completion of this whole process as shown in Figure 1, this whole algorithm iterates from the top with the new generation until there are no satisfying solutions or if we have reached a extremely close approximation that converged and is deemed the optimal solution.

2.5 Why use the Genetic Algorithm

Though briefly describing the GA and its capability to solve problems which cannot be solved in any other means, we thought it would be beneficial to describe a more in-depth reasoning on what makes the GA so great in computation:

- In parallel, GA's can process, populations that are billions times larger than a conventional computational algorithm. Particularly the expectation of a larger population is that it can sustain larger ranges of genetic variation, and thus generate high-fitness individuals in fewer generations.
- Stochastic optimization relies on a specific criterion, and in bioinformatics the GA is used as a more natural and appropriate tool to handle such criterion (e.g. alignment score, overlap strength, energy, etc.).
- Problems in bioinformatics need the exact optimum solutions, which GA's are typically well known for in order to require robust, fast, and approximatite solutions.

- Laboratory experiments and operations that involve DNA inherently involve errors. So using a deterministic algorithm where mutations occur, would highly break the characterstics of it being classified as deterministic, because given some particular input with mutations occuring, it may not always produce the same output in optimization problems. However with the GA, which is an evolutionary algorithm, mutations can contribute to genetic diversity, which is a desired property of such algorithms.

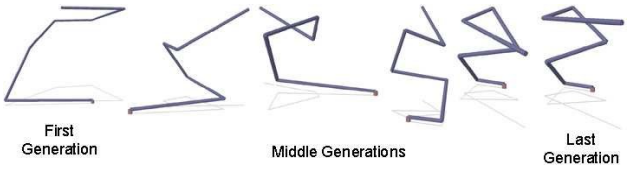

Figure 2: The series of evolutionary stages it took when working with the *evolved antenna* by *NASA*

After observing some advantages in using the Genetic Algorithm, we begin to explore how this method can be used to approach some application based problem. One of the most famous applications of the GA was used on the *evolved antenna* performed by *NASA*, which produced an optimized solution to see how an antenna could transmit the best signal through an evolutionary series of bends as shown in Figure 2. There have also been many occasions in bioinformatics where the alignment and comparisons of DNA, RNA, and protein sequences have been studied by the GA, as well as gene mappings, promoter identification, interpertations of micro array sequences, and many more. We wish to approach a similar problem in the field of bioinformatics where we begin to look at how this algorithm can be applied to our research, on the *Phylogenetic Tree Reconstruction*.

3 The Phylogenetic Tree

Phylogenetic Tree is an important representation constructed in many areas of biological research, ranging from systematic studies to the methods used for genome annotation [6]. This method has been used by bioinformaticians, to identify disease causing microbes, and determine its origin, how it might have spread, and routes of transmissions. In fact, the 2020 *Severe Acute Respiratory Syndrome Coronavirus 2* (SARS-CoV-2), and its variants can be well tracked using the tools from phylogenetics, where each viral nucleic sequence can be analyzed to determine how it mutated based off of both the parent sequences and in a broader scale the *SARS-CoV-1*. By identifying the sequences of these variants, they can then be classified to be more dangerous by its characterstics/behaviors and whether they can forgo the *mRNA* vaccines based on used by *Pfizer BioNTech* and *Moderna*.

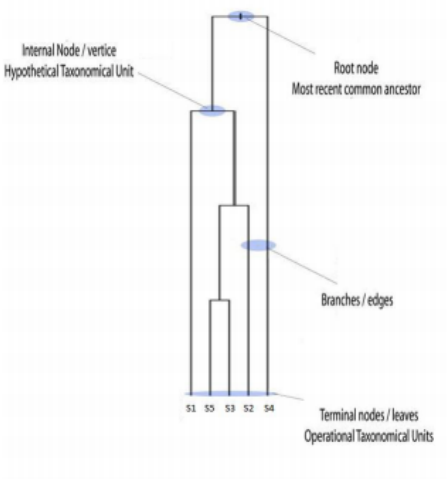

Figure 3: A visual representation of the *Phylogenetic Tree and its components* [5]

The basic components of a phylogenetic tree as represented by Figure 3, contains branches, also called edges that are connected to and terminate nodes or vertices. Branches can be classified in one of two ways: internal or external (terminal). These terminal nodes that are found at the tips of trees represent *operational taxonomic units* also reffered to as OTU's. The essence of an OTU is to correspond their molecular sequences or species (taxa) from which the tree is deduced. Conversely, internal nodes represent the last common ancestor to the nodes that arise from the that point. The type of tree, we intend to work with is composed of a multi-gene family called a gene tree which looks at one species and all its different variations.

3.1 The NP-hard Problem

Finding the most fit tree (solution) under any criterion, is considered to be a NP-hard problem within phylogenetic trees. When trying to construct a tree to find optimized solutions, it is best to use a heuristic that can both be classified as a search and optimization algorithm. That is why the GA is a great tool for such problems being a metaheuristic that specializes in navigating large search spaces to find optimized solutions in stochastic systems. Research has previously been done on GA with the *subtree-pruning-and-regrafting* (SPR) and *nearest-neighbor-interchange* (NNI) methods [6], in which it was proven to show that the SPR was able to reproduce very close optimized solutions that were not too significantly off to that of the best tree. In our case, we wish to utilize the *neighbor-joining algorithm*, which feeds off a distance-matrix produced by the a multiple sequence alignment algorithm that uses the GA called *Sequence alignment by genetic algorithm* (SAGA) that shows the relationship between closely related protein sequences needed to construct our tree for our Deltacoronavirus data from the NCBI. With this dataset, we intend to model the relationships of the SARS-CoV-2 Deltacoronavirus as a phylogenetic tree between 10 different species DNA genomes which will then be transformed into protein sequences, in which we will produce the most optimized solution to find out the patterns of evolution labeled by their genbank access codes. Before we begin to talk about our methodology, we wish to explain the more popular approach to the phylogenetic tree problem known as the Maximum-Likelihood Approach (ML).

3.2 The Maximum-Likelihood Approach

One of the slower (still fast for an NP-hard problem) but more frequent methods in tree reconstruction is that of the Maximum-Likelihood (ML) approach. This method is a statistical model of evolution where for each nucleotide position in a sequence, it approximates the probability of that position being a particular nucleotide, based on whether the internal node possess that specific nucleotide. The probabilities are then calculated for the branches of the bifurcating tree. ML is based on the idea that each nucleotide grows independently, enabling the phylogenetic relationship to be analayzed at each site.

Mathematically we can visualize this method with the Cavendar-Farris model [8]

$$\tilde{L}(\chi;T,\mathbf{p}) \cong -ln2\mathbb{P}[\chi|T,\mathbf{p}] = -ln(\sum_{\tilde{\chi}\in H(\chi)}\prod_{e=(u,v)\in E(T)} p_e^{1\{\hat{\chi}(u)\neq\hat{\chi}(v)\}}(1-p_e)^{1\{\hat{\chi}(u)=\hat{\chi}(v)\}}) \tag{1}$$

So suppose we are given some arbitrary tree T on n leaves and the probabilities of transition on edges is, $\mathbf{p} = \{p_e\}_{e\in E(T)} \in [0,1/2]^{E_T}$, where E(T) is the set of edges for the tree and $E(T) \cong |E(T)|$ is the cardinality of E(T) [8]. An understanding for the model can be obtained from the following steps: choose any vertex as the root; pick a state {0,1} for the root uniformly at random; When moving farther away from the root, each edge e flips the state of its ancestor with the given probability p_e. And lastly we have to also assign a state for the leaves using the character χ. An extension of χ occurs when the the vertices of T are assigned a state of {0,1} for those equal to χ on the leaves. This set of all extensions χ can be reffered to $H(\chi)$.

Now that the fundamental componenets are understood, let us also explain what the Equation 1's, the Cavendar-Farris model purpose is. In this model if the set A is 1, it is known that event A occurred, and would be 0 otherwise. In the maximum likelihood problem, we wish to compute $(T*,\mathbf{p}*)$, which is minimizing $\tilde{L}(\chi;T,\mathbf{p})$ over all trees and transition probability vectors. So by computing all the sites, we can then assess which tree has the best fitness which will be deemed as our optimal solution. But as can seen, its inefficiencies lie directly because it constructs all nucleotides independently, and constructing and analyzing every single site can be computationally slow. But regardless of speed, plenty of research has been done using the Maximum Likelihood method with the Genetic Algorithm, and to this day, it is probably the most popular to solve phylogenetic tree reconstruction NP-problems due to its accuracy.

3.3 The Distance-Matrix Approach

Our preferred method of choice and the focus of this paper will however be on the *Distance-matrix Method*. Distance-matrix methods are far more rapid compared to the maximum-likelihood, and it essentially measures the genetic distances between sequences. Once these distances are found, a distance matrix will be constructed, where the distance equates to the approximate evolutionary distance. However, the challenging aspect of this computational method is constructing the matrix itself. The distance matrix can be visualized as a $n * n$ matrix, where n resembles the number of sequences. Every row corresponds to a single sequence and every columns corresponds to the distance between the sequences. By getting some collection of n sequences, the matrix below can be constructed:

$$\begin{bmatrix} d_{11} & \cdots & d_{1n} \\ d_{21} & d_{22} & d_{2n} \\ \vdots & \ddots & \vdots \\ d_{n1} & \cdots & d_{nn} \end{bmatrix} \tag{2}$$

In the distance matrix above, d_{ij} can be broken down to where the distance is measured between the i_{th} and j_{th} sequences. Though this measure of weight is up to choice with methods like *Jukes-Canton and p-Distances*, we are going to be using a popular multiple sequence alignment method called *Sequence Alignment by Genetic Algorithm* (SAGA), because we are working with 3 or more genomes at once. This method will be used to obtain the genetic distance from a sequence alignment to help us construct our distance matrix with additional assistance from the GA. The algorithm will be explained in greater detail in Section 4.1. We will also be introducing the *neighbor-joining algorithm* presented in Section 4.2 which uses the distance-matrix produced by SAGA to compute an unresolved star tree topology, and transforms it into a tree where all the branches lengths are known. This method relies solely on how the matrix is constructed because there are a number of combinations that can be constructed within the distance matrices to feed the algorithm.

Distance-based methods like the neighbor-joining algorithm, are polynomial time and are quite rapid in practice. Within the distance-based methods, there are two different algorithms: *cluster-based* and *optimality-based*. The cluster-based algorithm builds its distance matrix by starting at similar sequence pairs, which are then used to construct the tree. This algorithm is found within the *UPGMA* or *unweighted pair group method using arithmetic average*, and our *neighbor joining*. Meanwhile the optimality-based algorithm compare multitudes of different tree topologies and selects the best fit by determining computing distances in the trees, and the desired evolution or *actual evolutionary distance*. This algorithm is common in *Fitch-Margoliash* and *minimum evolution* methods. But in retrospect, both these algorithms are computationally simpler than ML and therefore have our interests based on the advantage of speed.

4 Our Methodology

In this section we now wish to bring in all the different components described in the previous sections to begin seeing how all the pieces fit together. In essence, we wish to construct phylogenetic trees to model the spread of the SARS-CoV-2 Deltacoronavirus epidemic between 10 different species DNA sequences. We identified in the previous section that we wish to approach the phylogenetic tree problem, using the Neighbor-joining (NJ) algorithm, which is fed a distance matrix built by the Sequence alignment by genetic algorithm (SAGA) (*SAGA only works on protein sequences, so DNA must be converted into proteins) that compares and outputs the dissimilarities between all genome sequences.

4.1 Sequence Alignment by Genetic Algorithm

Because we are looking at more than 3 different sequences of the SARS-CoV-2 Deltacoronviruses which changed overtime, a multiple sequence alignment like SAGA is a very practical tool used in bioinformatics to help analyze these dissimilarities within multiple protein sequences. Because we have raw DNA data from the NCBI, we first must convert the DNA sequences into amino acids, and transform them into their protein forms. However, when SAGA is computed, because there will be less differences from amino acids naturally compared to its DNA form, it will help be compared at this level more easily. Additionally, what makes this method better than mutliple sequence alignment (MSA) algorithms like MUSCLE (MUltiple Sequence Comparison by Log-Expectation) would be that it would take into account the genetic algorithm events like fitness, single-point crossovers, mutations

under certain criterion to build a single optimal tree in the form of a distance-matirx versus all the other methods which required a larger amount of computation. This approach to build out a distance matrix, is far easier to be analyzed once the algorithm is completed and is therefore more organized.

By obtaining the genetic distances which are the differences in sequence alignments from genome to genome from the SAGA software, we can then proceed to construct a distance-matrix. As a result, various tree topologies are visualized because not all trees are built in just one way, and can usually have multiple representations of the same distance-matrix. But thanks to SAGA, we have now prepared all the components for the tree construction itself by finalizing the matrix that we will feed into our MatLab neighbor-joining programs. So with this information, we can use it in our SARS Deltacoronavirus research to find out the pattern of evolution based on the different genome samples from the NCBI database.

4.2 Neighbor-joining Algorithm

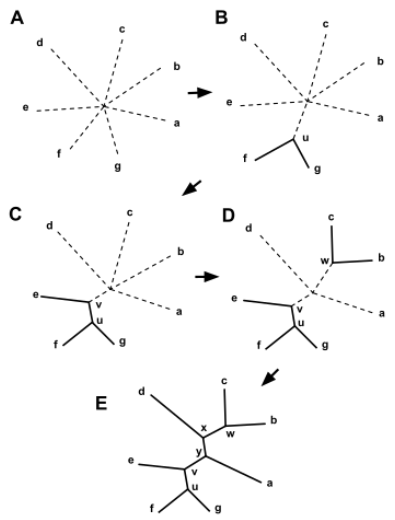

Figure 4: A visual representation of a distance matrix containing 7 species, beginning in the star-shape which then is then slowly constructed into a tree using the neighbor-joining algorithm based off a specific distance-matrix

Now that we have obtained a distance matrix, we can begin to construct the trees. As described in Figure 4, the tree begins in a star formation where there is no tree or lengths of branches shown or combined into internal nodes. But with this distance matrix, the distances represent the dissimilarity of the aligned sequence and can begin subsequent joining between neighbors. So from the diagram we can see that each OTU is represented as a fork. For each OTU, you want to compute the sum of distance between OTU one and another OTU, divided by (N-2), where N is the total number of OTU's. The equation can be represented to describe the following:

$$S_i = \frac{1}{N-2} \sum_{k=1}^{N} D_{ik} \tag{3}$$

Once you combine the two taxa on a node into a sub-tree, you want to calculate the new branch length which can be calculated by the following equation:

$$D_{Xi} = \frac{D_{ij} + S_i - S_j}{2}, D_{Xj} = \frac{D_{ij} + S_j - S_i}{2} \tag{4}$$

And at last calculating the new matrix distance is done by connecting i and j and replace it with a node x that connects it to the final equation:

$$D_{xk} = \frac{D_{ik} + D_{jk} - D_{ij}}{2} \tag{5}$$

This sequence of steps are repeated for (N-2)! iterations.

5 Our Analysis and Findings

At last, we will begin discussing our analysis, and findings of the SARS-CoV-2 Deltacoronavirus and how it spread amonst the 10 species sequences offered by the NCBI. As described in Section 4.1, because we were given raw data of DNA genomes, we first must convert them to its protein sequence form. In addition to the transformation from a DNA to a protein, the first steps is to transcribe DNA into RNA. Luckily this is quite simple in that all nucleotide bases remain the same minus the thymine (T) nucleotide base, which is converted into Uracil (U). Once in its RNA form, you translate the mRNA which selects codons or 3-pair RNA sequences and builds subunits of the proteins that contain a specific series of amino acids in which you will end up with the protein itself. We perform this computation through a python module in simple fashion as shown in the table below.

Genbank Code	Polyprotein Sequence (first 25 Amino Acids)
HKU11-934	MVKNVSKRSPIVLPQIQPPPLQLFI
HKU12-600	MAMNIAKRSPIVLPQIQPPPLQLFI
HKU13-3514	MAKNKEKRSPIVLPELVPPPLQLFI
HKU16	MGKNNPKRSPIVLPDPIPPPLQLFI
HKU17	MAKNKSKRDAIALPENVPPPLQLFI
HKU18	MAKNKEKRSPIALPEVVPPPLQLFI
HKU19	MGSKQVDHTCLTIPPNPSKTLALFI
HKU20	MANKARPKGILVIPELSNNSSLLLL
HKU21	MTKNSFDVGKVITLPKVIPPPLQLF
HKU15	MAKNKSKRDAIALPENVPPPLQLFI

In the above table, we are working specifically with the polyproteins which show early signs of similarities. Within these similarities are slight differences which will all be accounted for when the SAGA takes place. But through conducting some initial analysis the similar protein sequences will be joining together first when the neighbor-joining algorithm takes place, and we can begin to see some correlations between sequences.

References

[1] Fortnow, L., *The Status of the P versus NP problem*, September 2009, Communications of the ACM 52(9):78-86

[2] Ahn, C.W., Ramakrishna, R.S., *Elitism-based compact genetic algorithms*, IEEE Transactions on Evolutionary Computation, Volume: 7, Issue: 4, Aug. 2003, Pages 367-385.

[3] Gammaitoni L., Hänggi P., Jung P., and Marchesoni F., *Stochastic Resonance* Rev. Mod. Phys. 70, 223–288 (1998)

[4] Gutierrez-Navaro, D., Lopez-Aguayo, S., *Solving ordinary differential equations using genetic algorithms and the Taylor series matrix method*, 2018 J. Phys. Commun. 2 115010

[5] Gupta, M., Singh S,. *A Novel Genetic Algorithm based Approach for Optimization of Distance Matrix for Phylogenetic Tree Construction*, International Journal of Computer Applications (0975 – 8887)

[6] Money, D., Whelan, S., *Characterizing the Phylogenetic Tree-Search Problem*, Systematic Biology, Volume 61, Issue 2, March 2012, Page 228.

[7] Eiben A.E., Raué P.E., Ruttkay Z. (1994) *Genetic algorithms with multi-parent recombination In*: Davidor Y., Schwefel HP., Männer R. (eds) Parallel Problem Solving from Nature — PPSN III. PPSN 1994. Lecture Notes in Computer Science, vol 866. Springer, Berlin, Heidelberg.

[8] Roch, S., *A Short Proof that Phylogenetic Tree Reconstruction by Maximum Likelihood Is Hard*, IEEE/ACM Transactions on Computational Biology and Bioinformatics, Volume 3 Issue 1, January 2006

[9] Tuffley, C., Steel, M., *Links between maximum likelihood and maximum parsimony under a simple model of site substitution*, Bull Math Biol., vol. 59, no. 3, pp. 581-607, 1997.

[10] Madeira F, Park YM, Lee J, et al. *The EMBL-EBI search and sequence analysis tools APIs in 2019*. Nucleic Acids Research. 2019