Repliclade: A Simulator of Sequence Evolution in Gene Families

by

Haki Dehari

A thesis submitted in partial fulfillment of
the requirements for the degree of

Master of Science

Computer Science

At

The University of Wisconsin–Whitewater

May, 2021

Graduate Studies

The members of the Committee approve the thesis of
Haki Dehari presented on April 30th, 2021

Dr. Robert Kuzoff, Chair

Dr. Arnab Ganguly

Dr. Christopher Veldkamp

ACKNOWLEDGMENTS

I would like to first give a large debt of gratitude to my thesis adviser, Professor Robert Kuzoff. Without his guidance and knowledge of the subject matter, this thesis would not have been possible. Dr. Kuzoff's love for the field of Computer Science and Biology helped define and shape my entrance into the Bioinformatics world and helped breathe life into my research. For that, I will forever be grateful.

I would also like to thank the rest of the thesis defense committee members, Dr. Arnab Ganguly and Dr. Christopher Veldkamp, whose feedback was paramount in the completion of the final product of the thesis. Another debt of gratitude is also owed to the UW-Whitewater Computer Science Department, whose teachings throughout the years during undergraduate and graduate school have helped mould me into a true scholar and researcher.

Last but not least, I would like to thank my wife and parents for their unwavering support as I worked towards a Master's degree while working full time. Through both the easier and more difficult times, their support was always a constant. Without their love and support, none of this would have been possible.

Repliclade: A Simulator of Sequence Evolution in Gene Families

By

Haki Dehari

The University of Wisconsin-Whitewater, 2021

Under the Supervision of Dr. Robert Kuzoff

An ever-increasing need in phylogenetic inference entails selection of suitable analytic methods for a given lineage. It also includes identifying appropriate evolutionary models to perform forwards (or backwards) simulation on that lineage. These analytic methods and evolutionary models are prone to error and are known to bias phylogenetic simulation results. This can lead to erroneous results in an inferred phylogeny or phyletic simulation. It is of paramount importance to allow for selection of multiple methods of sequence simulation to perform phylogenetic inference with minimal prejudice. In this thesis, I present Repliclade. Repliclade is a software package written in the Python programming language that utilizes statistical methods to evaluate and compare models of molecular evolution and estimate values for associated parameters to facilitate robust phylogenetic inference on a monophyletic lineage of interest. It will then execute a simulation of a single inferred ancestral sequence generated via the Coalescent Theory once various crucial sequence parameters have been extracted. A phylogenetic reconstruction of the derived sequences are generated as a result of the simulation. The resulting progeny sequences and performance metrics along with numerous methods of phylogenetic reconstruction will provide insights into the efficacy of competing evolutionary models and methods of inference implemented by other software packages. These insights will assist

researchers in identifying the strengths and weaknesses of varying approaches to simulation of an ancestral sequence and phylogenetic estimation. By corollary, Repliclade addresses an important and currently unmet need in the world of phylogenetic inference for informed model selection and sequence parameter extraction during DNA sequence analysis.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

The need for accuracy in phylogenetic estimation is monumental. Mathematical models of evolution tend to be more straightforward when compared to the realistic underlying mechanisms of evolution. Diverse methods of phylogenetic inference and simulation have been proposed over the years to try to minimize these errors. Unsurprisingly, there are instances where biologists and geneticists are unsure of which evolutionary model to use to infer progeny of an ancestral genome. There are other cases where a biologist needs to derive specific values which they will find useful in the field of molecular population genetics but are unable to due to a lack of programming knowledge or suitable software to determine such values. Repliclade, a simulator of sequence evolution in gene families, has been developed to address this unmet need in the biological community. This new software package gives the user the ability to simulate the evolution of an ancestral sequence and explore the suitability of competing evolutionary models for analysis of a biological lineage of interest. Repliclade also extracts crucial sequence parameters used to infer progeny sequences and provides estimates of other important values such as coalescence time. There has been previous work, such as PhyML (Guindon et al., 2003), which utilized different models of molecular evolution-should the user so choose-and uses the specified model to estimate maximum-likelihood phylogenies from nucleotide or amino acid sequence alignments. While PhyML is revolutionary in its own right, but it does not fulfill the need for flexibility to determine which models are most suitable for

the simulation of nucleotide sequence evolution. It instead infers phylogenetic trees using cutting edge maximum-likelihood methods to determine the most probable phylogeny for a set of input sequences. Repliclade is similar in this regard, where it takes an input sequence and utilizes it for its calculations.

The underlying mechanisms of Repliclade can be categorized into three different components. The first of these components is the alignment of a set of sequences that are homologous to the input sequence. Repliclade takes an input sequence provided by the user, and queries the input sequence within GenBank. GenBank is a comprehensive database that contains hundreds of thousands of nucleotide sequences. Repliclade utilizes the BLAST algorithm (Altschul et al., 1990) to retrieve a list of sequences which are similar to the input sequence. Once these similar nucleotide sequences are retrieved, Repliclade will align the sequences using a multiple sequence alignment tool called MUSCLE (Edgar et al., 2004). MUSCLE will return a list of aligned sequences which will then be used by Repliclade's second main component.

The second main component of Repliclade is DNA sequence parameter extraction. Depending on the initial input sequence, Repliclade will first retrieve sequences that are homologous to the original, and then generate a global alignment of all input sequences. Multiple sequence alignment is an integral part of Repliclade, since these alignments can then be used to extract various population parameters which will be crucial to the inference of an ancestral sequence. Parameters extracted by Repliclade include the effective population size (Wright, 1931) and genetic diversity (Li et al., 1979).

The third component of Repliclade is the simulator. After inferring an ancestral sequence via a process called The Coalescent (Kingman et al., 1980), Repliclade then allows the user to select from a list an evolutionary model to employ in ancestral sequence inference. Additionally, it allows the user to choose a desired simulation time (in generations). With each generation, Repliclade performs a plethora of calculations in the background to derive probabilities of evolutionary operations such as SNP's (single nucleotide polymorphisms) or duplication events. Once the running time has been exhausted, Repliclade will write the sequences which

existed throughout the simulation to a file to facilitate comparison of results from varying generations and different points of time throughout the simulation.

1.1 Evolutionary Models

During the simulation, Repliclade incorporates evolutionary models when calculating probabilities of mutations. An evolutionary model is a mathematical model based on Continuous Time Markov Chains (Markov, 1906) or better known as Markov Models. A Markov model is a stochastic model that is used to model continuously changing systems. These systems are most commonly represented as two-dimensional matrices and have important uses in the field of bioinformatics as well as machine learning and artificial intelligence. Repliclade utilizes these models to ascertain probabilities of mutation and to keep track of the elapsed time (in generations) of a chosen model which is associated with a particular nucleotide sequence in the simulation. During execution of the Markov chain, the probability that the current nucleotide (pre-mutation) will stay the same is held constant. This will be more evident in the subsections where we can see the difference in probability values between a nucleotide staying the same versus the nucleotide mutating. The user is prompted to select among a set of available evolutionary models before beginning the simulation. Thereafter, Repliclade records running estimates for parameter values as the simulation progresses.

Evolutionary models store probabilities in a mathematical structure called a stochastic or transition matrix (Markov, 1908). These structures are square matrices where each element of the matrix is a probability corresponding to the row and column indices of the matrix. The row and column indices in a nucleotide substitution model as used by Repliclade correspond to the 4 nucleotides A, T, C and G. The probability calculations differ by evolutionary model yet the structure for the transition matrices do not change. When a probability threshold is met, a single nucleotide polymorphism (SNP) occurs. A SNP is a disparity of nucleotides present at a single position in a multiple sequence alignment. Although there are other types of mutation in DNA, the evolutionary models currently implemented in Repliclade account for base substitutions, alone. SNP's can be conceptually divided into two different groups, transitions and

transversions. A transition is when a purine (A or G) mutates to another purine or a pyrimidine (C or T) mutates to another pyrimidine. A transversion is when a purine mutates to a pyrimidine or vice versa. In general, transitions (Ti) are much more common than transversions (Tv), typically outnumbering the latter by at least two to one (Yang et al., 1999).

1.1.1 Jukes and Cantor 1969 (JC69)

One of the most common models of molecular evolution is the Jukes and Cantor 1969 (JC69) model. It is widely used in evolutionary analyses, owing to its remarkable efficiency in computational analyses. It assumes equal base frequency for each nucleotide such that

$$\pi_A = \pi_G = \pi_C = \pi_T = 1/4 \tag{1.1}$$

JC69 also assumes that the mutation rate is the same across all nucleotides and does not differentiate between transitions and transversions as some other evolutionary models do (Kimura, 1980). The probability matrix of the JC69 model can be written as follows:

$$P = \begin{bmatrix} \frac{1}{4} + \frac{3}{4}e^{-t\mu} & \frac{1}{4} - \frac{1}{4}e^{-t\mu} & \frac{1}{4} - \frac{1}{4}e^{-t\mu} & \frac{1}{4} - \frac{1}{4}e^{-t\mu} \\ \frac{1}{4} - \frac{1}{4}e^{-t\mu} & \frac{1}{4} + \frac{3}{4}e^{-t\mu} & \frac{1}{4} - \frac{1}{4}e^{-t\mu} & \frac{1}{4} - \frac{1}{4}e^{-t\mu} \\ \frac{1}{4} - \frac{1}{4}e^{-t\mu} & \frac{1}{4} - \frac{1}{4}e^{-t\mu} & \frac{1}{4} + \frac{3}{4}e^{-t\mu} & \frac{1}{4} - \frac{1}{4}e^{-t\mu} \\ \frac{1}{4} - \frac{1}{4}e^{-t\mu} & \frac{1}{4} - \frac{1}{4}e^{-t\mu} & \frac{1}{4} - \frac{1}{4}e^{-t\mu} & \frac{1}{4} + \frac{3}{4}e^{-t\mu} \end{bmatrix} \tag{1.2}$$

where t is the time variable that changes with each iteration of the simulation and μ denotes the mutation rate of that particular sequence. As explained above, the mutation rate for the JC69 model is constant across all types of SNP's. It does not differentiate between transitions or transversions for this particular model and opts instead for simplicity when performing the calculations. The JC69 model also has a method of estimating evolutionary distance between two sequences which is derived via the following equation:

$$\hat{d} = -\frac{3}{4} ln(1 - \frac{4}{3}p) \tag{1.3}$$

where p is the number of positions that the two sequences differ. Importantly, the JC69 model does not require many calculations and substantial computational power. However, the exceptional efficiency of JC69 often comes at the expense of accuracy, especially in lineages with pronounced Ti:Tv ratios. There are other models which tend to account for different evolutionary forces and statistical differences which exist in nature.

1.1.2 Kimura 2-Parameter Model (K2P)

Kimura's two parameter model (K2P) (Kimura, 1980) differentiates between the instantaneous rates of transitions and transversions when calculating the probabilities of mutations via a Markov Model. As described in the preceding subsection, the JC69 model does not account for rate disparities between transitions and transversions and instead uses a single value for the mutation rate. Kimura's 2P model more accurately reflects the real world dynamics of sequence evolution such that transversions are associated with a lower probability of occurrence relative to transitions. Because of this, the K2P model is generally accepted as a more accurate model than JC69. The trade-off is that the model now becomes more computationally expensive. The K2P model also assumes equivalent base frequencies for all nucleotides similar to the JC69 model. Kimura derived the probability of a transition via the following formula:

$$P_{transition} = \frac{1}{4} + \frac{1}{4}e^{-4t\beta} - \frac{1}{2}e^{-2(\alpha+\beta)t} \tag{1.4}$$

where β represents the transversion rate and α represents the transition rate. Kimura also derived the probability of a transversion which is represented by the following equation:

$$P_{transversion} = \frac{1}{4} - \frac{1}{4}e^{-4\beta t} \tag{1.5}$$

There is also a way to calculate the genetic distance between two sequences simulated by the K2P model similar to the JC69 model. The calculation to determine genetic distance is as follows:

$$\hat{d} = -\frac{1}{2} ln((1 - 2p - q)\sqrt{1 - 2q}) \tag{1.6}$$

where p is denoted as the number of transitions between the two compared sequences and q is denoted as the number of transversions.

1.1.3 Kimura 3-Parameter Model (K3P)

An extension of the K2P model is the Kimura 3-Parameter Model (Kimura, 1981). In the K2P model, Kimura used two distinct values for transitions and transversions. In the K3P model, Kimura uses the same distinct value for transitions and two distinct values for transversions, introducing a new variable γ, which denotes the conservation of the weak/strong hydrogen-bonding properties of nucleotides. In short, this would represent the following transversions:

$$A \longleftrightarrow T \tag{1.7}$$

and

$$C \longleftrightarrow G \tag{1.8}$$

The other type of transversion is represented by β which denotes the amino/keto properties of nucleotides. This would represent the remaining following transversions:

$$A \longleftrightarrow C \tag{1.9}$$

and

$$G \longleftrightarrow T \tag{1.10}$$

The K3P model is more seldom used than the K2P model overall. One important property of the K3P model is the ability to perform a technique called a Hadamard Transform assuming that a phylogeny was generated on a tree where the taxa were evolved using the K3P model. A Hadamard Transform (Hadamard et al.) is a type of Fourier Transform (Fourier, 1822) which performs an orthogonal transform on a matrix.

1.1.4 Felsenstein 1981 Model (F81)

Another evolutionary model is the Felsenstein 1981 Model (Felsenstein, 1981). Unlike JC69 and K2P, the F81 model accounts for varying base frequencies, which are common in naturally occurring biological lineages. Base frequency is the proportion of occurrences of a nucleotide relative to the whole DNA sequence or to a group of DNA sequences. As previously stated the K2P and JC69 models assume static base frequencies of $\frac{1}{4}$ for each nucleotide base. In contrast, the F81 model allows base frequencies to vary for the probability calculations of mutations. This means that

$$\pi_A \neq \pi_G \neq \pi_C \neq \pi_T \neq \frac{1}{4} \tag{1.11}$$

One important aspect of the probability calculations is the β value. This is derived as

$$\beta = \frac{1}{1 - \pi_A^2 - \pi_G^2 - \pi_C^2 - \pi_T^2} \tag{1.12}$$

Therefore, the probabilities are derived differently and become

$$P_{ij}(t) = e^{-\beta t} + \pi j(1 - e^{-\beta t}) \tag{1.13}$$

if $i = j$ and

$$P_{ij}(t) = \pi_j(1 - e^{-\beta t}) \tag{1.14}$$

if $i \neq j$.

Due to the nature of varying base frequencies, this evolutionary model tends to make the nucleotide with the highest frequency more frequent as the simulation progresses through a substantial number of generations.

1.1.5 Hasegawa, Kishino, and Yano 1985 Model (HKY85)

After Felsenstein's F81 model, there were a few other population geneticists which followed his lead on using varying base frequencies in their probability calculations. Notable among them, are Hasegawa, Kishino, and Yano who derived the HKY85 model (Hasegawa, Kishino, Yano, 1985). The HKY85 model combined important innovations of both the F81 and K2P models. Specifically, the HKY85 model not only allowed for varying base frequencies, but also allowed varying rates for transitions and transversions as was first proposed by Kimura in the K2P model. Combining both of these important attributes makes the HKY85 model robust. This model introduces a new parameter, K, which represents the proportional difference between transitions and transversions. The model also uses β as a parameter and is calculated as follows

$$\beta = \frac{1}{2(\pi_A + \pi_G)(\pi_C + \pi_T) + 2K[(\pi_A\pi_G) + (\pi_C\pi_T)]} \tag{1.15}$$

Once the β parameter is derived, the probabilities of nucleotide mutation can be calculated as follows for the base pair A

$$P_{AA}(t,K,\pi) = \frac{[\pi_A(\pi_A+\pi_G+(\pi_C+\pi_T)e^{-\beta t})+\pi_G e^{-(1+(\pi_A+\pi_G)(K-1))\beta t}]}{\pi_A+\pi_G} \tag{1.16}$$

$$P_{AC}(t,K,\pi) = \pi_C(1-e^{-\beta t}) \tag{1.17}$$

$$P_{AG}(t,K,\pi) = \frac{[\pi_G(\pi_A+\pi_G+(\pi_C+\pi_T)e^{-\beta t})-\pi_G e^{-(1+(\pi_A+\pi_G)(K-1))\beta t}]}{\pi_A+\pi_G} \tag{1.18}$$

$$P_{AT}(t,K,\pi) = \pi_T(1-e^{-\beta t}) \tag{1.19}$$

The same probability characteristics for the equations can be applied to all of the other nucleotide base pairs as well. Due to the calculations needed for the HKY85 model during each iteration of the simulation, we can assume that this model requires larger amounts of computational power than the other previously mentioned models. As a result, running the simulator in Repliclade using the HKY85 model tends to be computationally expensive.

1.1.6 Tamura 1992 Model (T92)

The Tamura 1992 (T92) Model is an extension to the Kimura 2-Parameter model. In the T92 model, the same two-parameter method is extended by considering the presence of GC content bias in the applicable DNA sequences. The GC content of a sequence is the proportion of nucleotide bases which are either G or C. The GC content is known to play an integral functional role within a genome. For example, there is evidence that the GC content of a coding region is directly proportional to the length of the full coding sequence (Pozzoli et al., 2008). The T92 model is particularly useful when dealing with sequences with strong transition and transversion biases as well as GC content biases. An example of this is the Drosophilia mitochondrial DNA (Tamura, 1992). In terms of the mathematics, the T92 model uses a single base frequency parameter such that:

$$\pi_{GC} = \pi_G + \pi_C = 1 - (\pi_A + \pi_T) \tag{1.20}$$

This means that all base frequencies for the nucleotides can be defined as a function of π_{GC}:

$$\pi_G = \pi_C = \frac{\pi_{GC}}{2} \tag{1.21}$$

and

$$\pi_A = \pi_T = \frac{1 - \pi_{GC}}{2} \tag{1.22}$$

Thus, the evolutionary distance between any two DNA sequences can be calculated by the following equation according to the T92 model:

$$d = -hln(1 - \frac{p}{h} - q) - \frac{1}{2}(1 - h)ln(1 - 2q) \tag{1.23}$$

where $h = 2\theta(1 - \theta)$ and θ is the GC content or π_{GC}. We can see that this distance calculation is similar to the K2P distance calculation in equation 1.6. The key difference is the T92 model is considering the GC content bias h along with the transition and transversion biases.

1.2 Sequence Parameter Extraction

DNA parameter extraction entails deriving critical parameter values from a group of DNA sequences. In the field of population genetics, researchers are able to generate these values

computationally using select statistical and mathematical methods that provide insights into the evolutionary dynamics of naturally occurring populations based on a set of related sequences sampled from representative individuals. While many of these parameter values can be estimated using observational methods, researchers do not always have the luxury of being able to use these methods. There is an excessive amount of information to process by human hands alone, and this is where computers play a major role in identifying patterns in genomic data that are difficult to identify with the human eye. Many DNA sequences are stored in large online databases. A prime example is GenBank. These databases can be used to retrieve groups of sequences with which computational methods can be applied to extract crucial values and inform researchers of the evolutionary forces shaping allelic diversity in a given population. Repliclade utilizes these varying parameters to assist the user in making an informed decision based on a group of DNA sequences.

1.2.1 Estimating the Effective Population Size

Repliclade employs the Effective Population size in a range of calculations. Effective Population size (N_e) is the number of individuals in a population which contribute offspring to the next generation. In the field of population genetics, N_e is a principal parameter for calculating critical values such as genetic variation. For example, it was recently shown that indel rates are directly proportional to the effective population size of a species (Michael Lynch et al., 2016). Depending on the parameters, one could estimate other values such as mutation rate (μ) through N_e. This is why it is such an integral part of Repliclade. Unfortunately, N_e is typically unknown without direct species or DNA sequence observation. Accordingly, Repliclade must use other means to estimate this important parameter. This consists of first calculating genetic sequence variation (θ). This is a required parameter due to the following equation for estimating genetic variation:

$$\theta = 4N_e\mu \tag{1.24}$$

Once this other parameter is calculated, only then are we be able to estimate an effective population size.

1.2.2 Calculating Genetic Variation (θ)

A well known dependency of estimating effective population size is genetic variation (θ) as mentioned in section 1.2.1. Without this value, there is no other way to extract effective population size computationally unless other values were known, such as the number of males or females in diploid organisms or the harmonic mean of overall population size through time. Since Repliclade does not have access to observational data for a specific input sequence, it must find another way to compute θ. We know that the genetic variation of a population is directly proportional to N_e (Ferreti et al., 2015) through the following equation

$$\theta = 4N_e\mu \tag{1.25}$$

where μ is the mutation rate for a particular population. Unfortunately, we do not have N_e at our disposal here, so we must use other estimators to assess the degree of genetic variation in a population. There are multiple estimators to do this, but Repliclade uses two specific methods which are more computationally tractable given the information at hand.

1.2.3 Watterson Estimator (θ_w)

A simple but elegant method to estimate θ, the genetic variation of a population, was proposed by Margaret Wu and G.A. Watterson in 1975 (Watterson et al., 1975). The method was coined the Watterson Estimator and is a fairly straightforward solution to the problem of estimating θ. The high level description is that θ is calculated by counting the number of segregating sites between a group of similar sequences. A segregating site is a site on the aligned group of sequences where a SNP exists. By quantifying the proportion of segregating sites in a group of sequences, we are able to get an intuitive sense of variation within the population.

We encounter a dilemma here where non-segregating sites may actually be segregating but are not observed when calculating the number of sites computationally. For example, let us say we have two DNA sequences which are exactly the same, ATGGC. If we compute the number of segregating sites between these two sequences, we may conclude that there are none because they are exactly the same. Now let us assume that at time x, the sequence was ATGGC as we mentioned previously. For the sake of the example, we can say that the sequence ATGGC now became ATGGT at time y. Finally, we can say that ATGGT became ATGGC at time z. If we were to now compare the two sequences at time x and at time z, we would calculate no segregating sites since the two sequences seem to be exactly alike. This is the problem we may encounter when using statistical methods such as the Watterson Estimator. Due to homoplasious back mutations at an individual site, we may calculate less segregating sites than there actually are in a group of DNA sequences. This is the trade-off we must make when utilizing statistical methods to extract sequence parameters. The Watterson Estimator relies on modeling, rather than observation alone, to account for these trade-offs. It can be calculated as follows:

$$\theta_w = \frac{K}{\alpha_n} \tag{1.26}$$

where K is the counted number of segregating sites and α_n is the $n-1$th harmonic number or:

$$\alpha_n = \sum_{i=1}^{n-1} \frac{1}{i} \tag{1.27}$$

where n is the sequence sample size. If the sequence sample size is only two, then the calculation for the number of segregating sites would be straightforward. Once the sample size becomes larger than two, we need to better define how a segregating site can be identified. A

specific threshold needs to be set to properly identify a segregating site in a group of DNA sequences. Implementation of the Watterson estimator in Repliclade will be further discussed in Chapter 2.

1.2.4 Fay and Wu Method for Genetic Variation (θ)

The Fay and Wu method for genetic variation estimation (Fay and Wu, 2000) uses ancestral state information to calculate a value for genetic variation. This means that the genetic variation can only be estimated if the ancestral sequence is known (or derived). Fay and Wu also utilize segregating sites in their calculations and use θ_H as their symbol to represent the genetic variation based on their method. The magnitude of θ_H determines whether segregating sites exist between the derived sequence and the ancestral sequence. By counting up these segregating sites and summing up the observed differences between a changing range of sequence numbers, Fay and Wu were able to estimate θ_H using their statistical method. θ_H can be calculated as follows:

$$\theta_H = \frac{\sum_{i=1}^{n-1} i^2 S_i}{n(n-1)/2} \tag{1.28}$$

where S_i is the number of segregating sites observed and i sequences carry the segregating site in comparison to the ancestral site. Therefore, it is imperative that the ancestral sequence is known before one can use the Fay and Wu method for their genetic variation estimation. How Repliclade derives an ancestral sequence will be discussed further in Chapter 2.

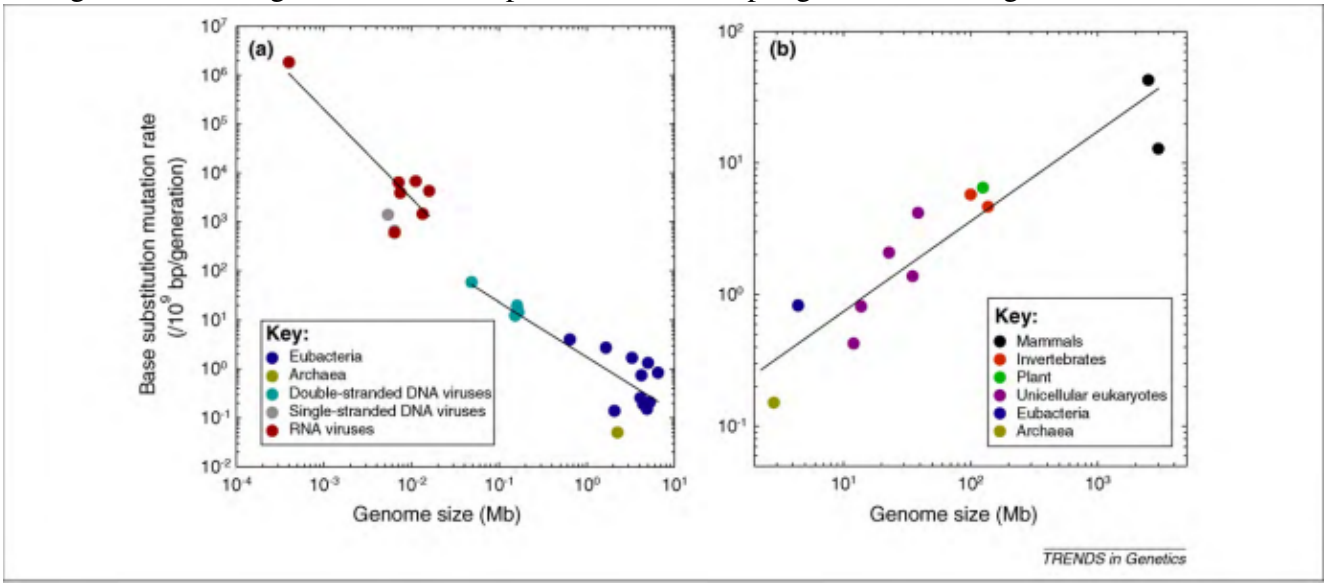
1.3 The Mutation Rate

Another key component of Repliclade and an integral parameter used for various calculations in the simulation is the mutation rate (usually denoted as μ). The definition of the

mutation rate is effectively the rate at which new mutations appear within a species or population. As noted in section 1.2.1, the mutation rate plays a direct role in estimating effective population size as well as genetic variation. To better understand μ, one must understand how it is defined and in what context it is used. Many researchers more properly define μ as the mutation rate "per nucleotide site per generation" (Lynch, 2010). This, put simply, means the rate at which each nucleotide site on a DNA sequence evolves or mutates.

There are a few statistics to describe mutation rate across various lifeforms. In double-stranded viruses and prokaryotes (single-celled organisms), we can see that the mutation rate scales inversely to the genome size of the organism.

Figure 1.1 Scaling of mutation rate per nucleotide site per generation with genome size



In comparison, mutation rates for eukaryotes (multicellular organisms) such as mammals and invertebrates tend to scale upwards as the genome size increases. This dissimilarity is commonly attributed to somatic mutation rates being higher than germline rates and germline mutation rates do not differ as much when compared to viral or prokaryotic genomes. As observed in figure 1.1, rates of mutations tend to differ dramatically between prokaryotes and eukaryotes. There are various statistical methods for estimating the mutation rate, but we have decided to leave this parameter value up to the user since those statistical methods are not

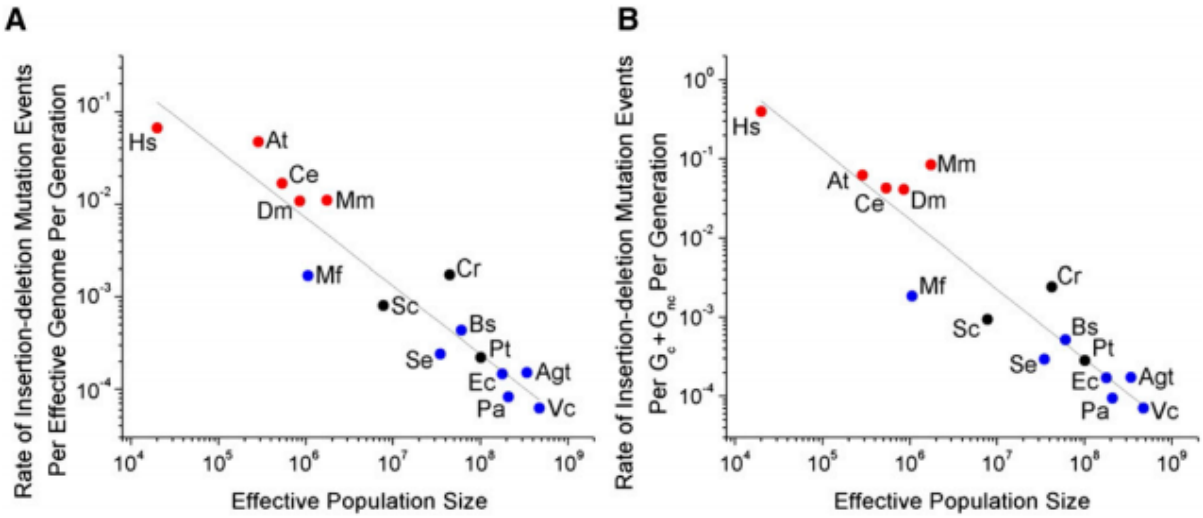
feasible with the current sequence information that Repliclade is able to generate or has at its disposal.

1.4 Indels

Thus far, we have only discussed single-nucleotide polymorphisms (SNP's) or also known as point mutations. Although this is the most commonly considered form of mutation, there are also other types of mutations that must be accounted for to accurately simulate genomic evolution. Another form of mutation is something called an indel. An indel is an insertion or deletion in the genome which occurs as a form of mutation. By insertion or deletion, this means that more nucleotides can get "pushed" into the DNA sequence or some nucleotides can get "pulled" out of the DNA sequence. Since nucleotides form groups of three base pairs called codons, which in turn code for amino acids, indels have a higher probability of disrupting the function of encoded proteins. Since indels can directly add or remove amino acids in the amino acid chains created from a genome, mutations of this variety are more likely to be deleterious. For example, indels are known to be the cause of various diseases such as cystic fibrosis and many different types of cancers (Gonzales et al., 2020). This is due to the fact that insertions and deletions tend to impact genomes negatively at a higher rate than SNP's.

Researchers have also shown that indel rates tend to decrease as the effective population size of a species or population increases (Lynch et al., 2016) (Fig 1.2). We can deduce this by measuring other parameters of a population and resolve certain characteristics of stable and abundant species. First we must define what a frame shift is in regards to a genome. We mentioned earlier that three groups of adjacent nucleotides that encode for an amino acid are termed codons. Since there are four different nucleotide bases, there are $4^3 = 64$ codons that can be formed from those four bases. Out of these 64 codons, there have been observed specific codons which signal the start and end of an open reading frame (the part of the DNA that has the ability to be translated into proteins), or ORF. When an indel occurs, it is more likely that a frame shift will result. A frame shift is a shift in the ORF which could potentially move the translation window for the DNA sequence. If the organism is lucky, this will be a neutral

Figure 1.2 Scaling of indel rates per effective genome per generation.



mutation (a mutation with no negative or positive effects on the genome). If the organism is unlucky, this frame shift on the ORF could lead to deleterious mutations and could impact the overall fitness of individuals in the population that bear this mutation. In populations with a large effective size, selection is the dominant force shaping allelic variation. Accordingly, indels, which are often deleterious, are less common when Ne is large. It is important to make this distinction about indels, since simulating them needs to mimic nature as closely as possible when these events occur computationally.

1.5 The Coalescent

In the subsection where we discussed genetic variation, it was noted that the Fay and Wu method for the estimation of θ required the knowledge of an ancestral sequence to the group of sequences in which θ was being calculated. A critical component of Repliclade is the ability to infer an ancestral sequence via a group of related sequences. This is also known as the Coalescent Process (Kingman, 1982). Commonly known as The Coalescent, this theory surmises that all variants of related sequences must have descended from one ancestral sequence at some

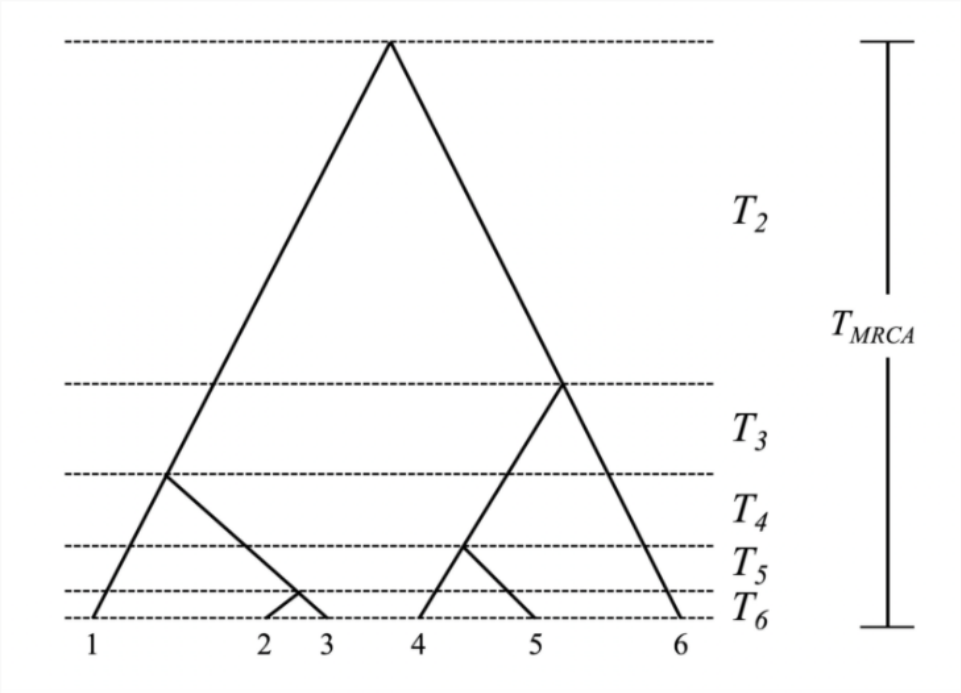
time in the distant past. One could say The Coalescent is a form of evolutionary model in which an ancestor must be inferred by looking backwards in time via a backwards simulation. On a higher level, the algorithm for the Coalescent is fairly straightforward.

1. Start with $i = n$ sequences.
2. Choose a time t from an exponential distribution, choose two sequences at random
3. Merge two sequences
4. Set $i = i - 1$.
5. Move to next time unit increment. Terminate if $i = 1$.

Although the internal workings of the Coalescent can get a bit more complex and is covered in Chapter 2, we can view some of the the inner workings of the algorithm from a high level explanation. The pairwise merging of sequences is paramount to the Coalescent process. A key assumption of the Coalescent it that the sample size n (the amount of sequences on hand) is much smaller than the effective population size (Takashi et al., 2003). If we violate this assumption, it can result in incorrect inferences about population processes. When utilizing evolutionary models discussed earlier, we may arrive at skewed results in the derivation of ancestral or progeny sequences. Historically, the different evolutionary forces which could be modeled using the Coalescent were limited. However, due to the exponential growth of processing power for computers, this has become less of an issue in recent years.

The Coalescent theory and algorithm play important roles in providing a framework for modeling the evolution of DNA sequences within lineages (Hahn, 2018). When the algorithm merges two sequences, it is theoretically merging two distinct genealogies. Since speciation is necessary for taxa to bifurcate, we must be able to reverse the speciation event and estimate how long ago this event took place. There are a few different calculations to take into account when simulating the Coalescent. We must first calculate the probability of a Coalescent event at time generation i. We can do this using the following equation:

Figure 1.3 A visual representation of the Coalescent algorithm.



$$P_{coalesce} = [(1 - \frac{1}{\binom{n}{2}2N_e})^{i-1}] * [\binom{n}{2}(\frac{1}{2N_e})] \tag{1.29}$$

where i is the unit generation time at which the probability is calculated and N_e is the effective population size as denoted in the earlier sections. These values are taken from a probability distribution at random when simulating the Coalescent. It is presumed that a population with a large effective population size has a higher degree of fitness than one with a lower population size. This is due to the fact that population sizes would directly correspond to a population's evolutionary fitness. The higher the effective population size, the more likely a species is able to contribute more offspring to the next generation as defined in the earlier sections. Theoretically, the effective population size is one of the main components of the Coalescent probability calculations.

There are also ways to roughly calculate how long ago the ancestral sequence existed for a group of sample DNA sequences. Let us remember that a Coalescent event occurs when two sequences chosen at random choose the same ancestor. When dealing with an effective population size of N_e, we can assume that the chance of one sequence choosing one particular ancestor out of this effective population size is $1/N_e$. Therefore, it is evident that the probability of two sequences choosing the same ancestor is half the previous calculation, or $1/2N_e$. Also, if we have a sample size of size n, the total amount of possible pairs would be equal to $\binom{n}{2}$. By multiplying the previous calculation with the total amount of possible pairs, we can see that another way to calculate the probability of a Coalescent event, or the probability of $n = i \rightarrow i - 1$ is to use the following equation:

$$P(i \rightarrow i-1) = \frac{i(i-1)}{4N_e} = \binom{i}{2}(\frac{1}{2N_e}) \tag{1.30}$$

where i is the current sample size of sequences which are left awaiting coalescence. We can now utilize the above equation to calculate the rough estimate for total Coalescence time. That is, the time it takes for i to become 1. We can approximate the Coalescence time for $i \rightarrow i - 1$ by an exponential distribution with a mean that is the inverse of the above equation (Hahn, 2018), which is:

$$E(T_i) = \frac{4N_e}{i(i-1)} \tag{1.31}$$

By selecting values at random from the exponential distribution to denote as T_i for the Coalescent time having i sequences left, we can conclude that the total Coalescence time for a group of n sample sequences can be roughly estimated as:

$$T = \sum_{i=2}^{n} T_i \tag{1.32}$$

The total Coalescence time gives valuable insight into the history of a sample size of DNA sequences and can be used to compare how the estimate fares in comparison to the actual simulated Coalescence time within a simulation. This is an important comparison to make when verifying the accuracy of the Coalescent simulation. One key point is that the rough estimate of the total Coalescent time should only be taken as a point estimate, and will most likely differ across varying sample sizes due to the randomized nature of the Coalescent process.

1.6 Phylogenetic Tree Reconstruction

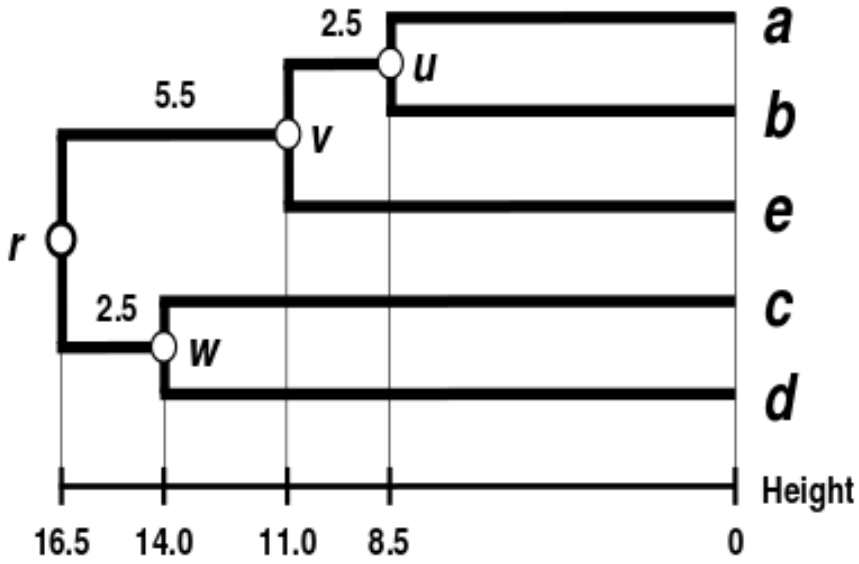
A phylogenetic tree is a tree diagram that represents evolutionary relationships between species. Building a visual representation of these relationships can empower scientists to better understand the ancestral history of a species or group of species. The branches in a phylogenetic tree intuitively represent how species diverged from a series of common ancestors. Two species are more closely related if they have a more recent common ancestor and more distantly related if they have a less recent common ancestor. Since there is usually no way to definitively know whether these evolutionary relationships among species in a phylogenetic tree are accurate, we must accept the fact that these trees are only hypotheses. Therefore, multiple methods of phylogenetic inference can be used to build trees from sequenced DNA which provide varying tree structures.

1.6.1 UPGMA

The Unweighted Pair Group Method with Arithmetic mean (UPGMA) phylogenetic tree building algorithm (Sokal et al., 1958) provides a bottom-up approach to clustering together

species on the tree. The algorithm constructs a rooted tree based on the pairwise distance matrices derived from the group of DNA sequences using a distance formula such as the distance equation for the JC69 model in section 1.1.1. It first finds the pair of sequences which have the least distance as calculated in the distance matrix. Once the shortest distance pair has been identified, it merges the 2 sequences into one node on the tree. These paired up sequences are now considered one node and the process continues grouping nodes and sequences together until all node tips are equidistant from the root as shown in figure 1.4.

Figure 1.4 A visual representation of a phylogenetic tree built using the UPGMA algorithm.



1.6.2 Neighbor-Joining

The Neighbor-Joining (NJ) algorithm is similar to the UPGMA method in that it also provides a bottom-up approach to building a phylogenetic tree. This algorithm also uses a distance matrix to derive the distances between each pair of genealogies. The main difference is that the NJ method starts with an unrooted and completely unresolved tree (Nei et al., 1987). It first finds the pair of sequences which have the least distance to each other. These sequences are

then joined into a single node which is connected to a central node. Then, the distance from each of the sequences in the pair to the new node is calculated. After that, the distance from each of the sequences outside of the new node to the new node are calculated. The aforementioned steps are then repeated after replacing the pair of joined neighbors with the new node. Ultimately, this algorithm results in an unrooted star-like tree as shown in figure 1.5.

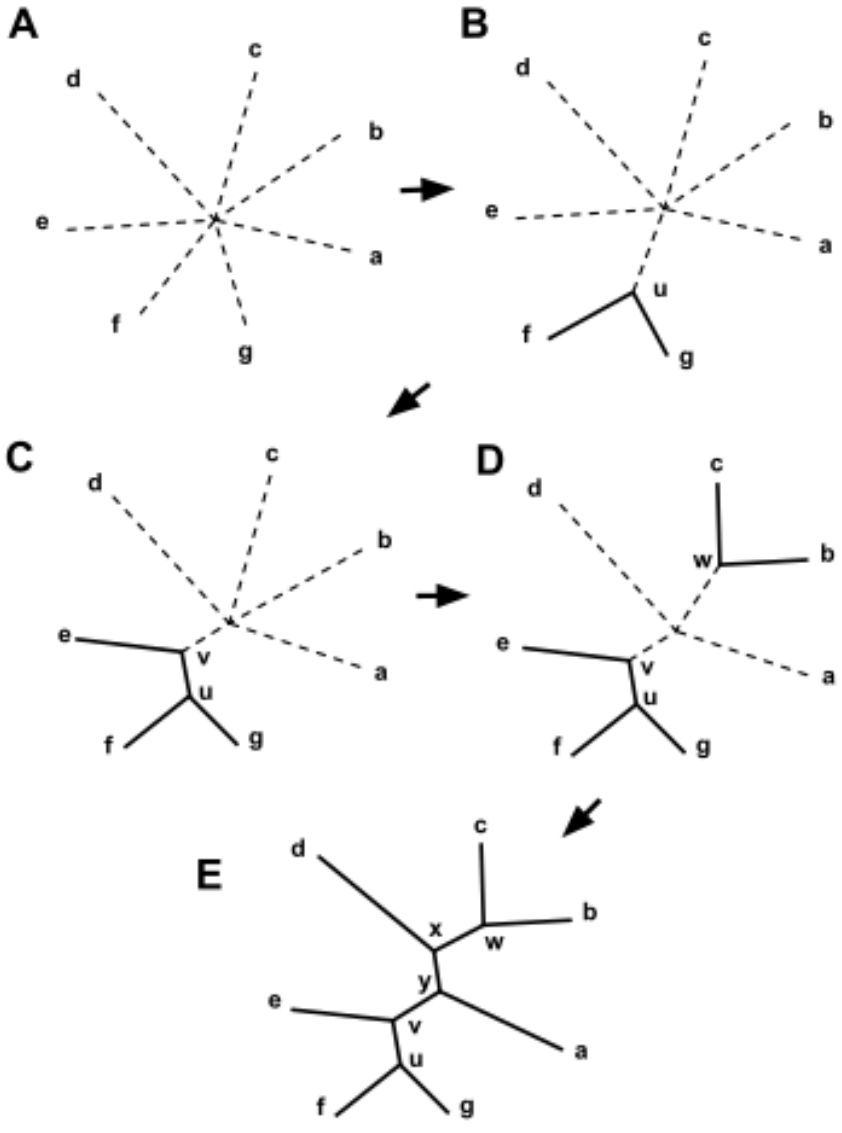
1.6.3 Maximum Parsimony

The Maximum Parsimony method is a character-based approach that creates a phylogenetic tree by minimizing the quantity of evolutionary changes needed to explain a given set of DNA sequence data (Farris, 1970). This means that the tree that explains the genomic data with the fewest number of character-state changes required is considered to be the best tree. There are three different types of searches that can be done to determine the best phylogenetic tree. There is an exhaustive search, which is guaranteed to find the best possible tree but is computationally expensive and is recommended only if there are less than 9 taxa involved. There is the branch-and-bound method, which is also guaranteed to find the globally optimal tree, but is only recommended when dealing with fewer than 20 taxa. When a greater number of sequences are involved, then one must use a heuristic search algorithm to infer a shortest phylogenetic tree, though it may represent a local optimum in solution space. As a heuristic method, this approach does not guarantee finding a globally optimal solution. However, it is generally the only tractable method when the number of taxa being compared exceeds 30, as is quite common.

1.6.4 Maximum Likelihood

The Maximum Likelihood phylogenetic reconstruction algorithm works by providing probabilities of sequences given an evolutionary model. The algorithm was introduced by Joseph Felsenstein (Felsenstein, 1981) and is one of the most used algorithms for phylogenetic reconstruction. The Maximum Likelihood method can become quickly intractable due to considering every possible tree for the highest overall probability score for that tree and is considered an

Figure 1.5 A visual representation of a phylogenetic tree built using the NJ algorithm.



NP-hard problem in Computer Science (Chor et al., 2004). There have been numerous heuristic algorithms proposed such as Felsenstein's pruning algorithm (Felsenstein, 2004) which reduces the overall search space of the Maximum Likelihood method. The PhyML program (Guindon et al., 2003) is utilized in Repliclade when inferring a phylogenetic tree using the Maximum Likelihood algorithm in Repliclade.

Chapter 2

Materials and Methods

2.1 Introduction

This chapter introduces a novel software package called Repliclade, which simulates genome evolution in a clade and informs the process of model selection during phylogeny estimation. Repliclade is a program which has been developed to consolidate many different types of phylogenetic analysis methods. Additionally, it provides the software user with the ability to determine which of a set of available phylogenetic analysis techniques are best suited for their user-specified DNA sequences. Repliclade has many different types of functionality which culminate in a simulation of a derived ancestral sequence generated from a pool of homologous sequences. Once the simulation completes, Repliclade will provide a list of phylogenetic tree reconstruction methods to the user to choose from to build a phylogenetic tree of the simulation results.

2.2 Initial Setup

Repliclade is written in the Python programming language and is compatible with any version of Python 3. Python is required to run this piece of software and the user can download Python at the following location: `https://www.python.org/downloads/`. After installing Python, a Python virtual environment needs to be created to install libraries that are required for the program. The proper libraries are contained in the *requirements.txt* file in the root program directory. More information on how to create virtual environemts and install

Python libraries can be accessed at the following location: `https://packaging.python.org/guides/installing-using-pip-and-virtual-environments/`.

2.3 Repliclade File System

Repliclade utilizes its own internal file system to work with files that are needed to process information that is required for the successful execution of the program. Repliclade generates files that are necessary for the later execution of critical subroutines. These files are crucial to analyses that Repliclade implements and are available for inspection by the user retrospectively to analyze values generated at distinct steps of the program life-span.

Figure 2.1 Repliclade file system

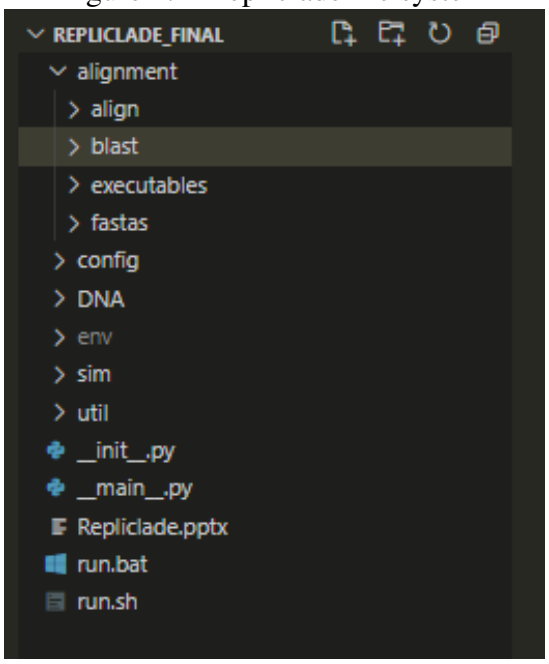


Figure 2.1 shows the general structure of the file system in Repliclade. When the user originally provides an input DNA sequence, the file must be in FASTA format and is stored to the *DNA* folder of the file system. The *FASTA* format is a commonly used file format in bioinformatics when dealing with DNA sequences. A thorough FASTA file specification can be found at the following location: `https://zhanglab.ccmb.med.umich.edu/FASTA/`.

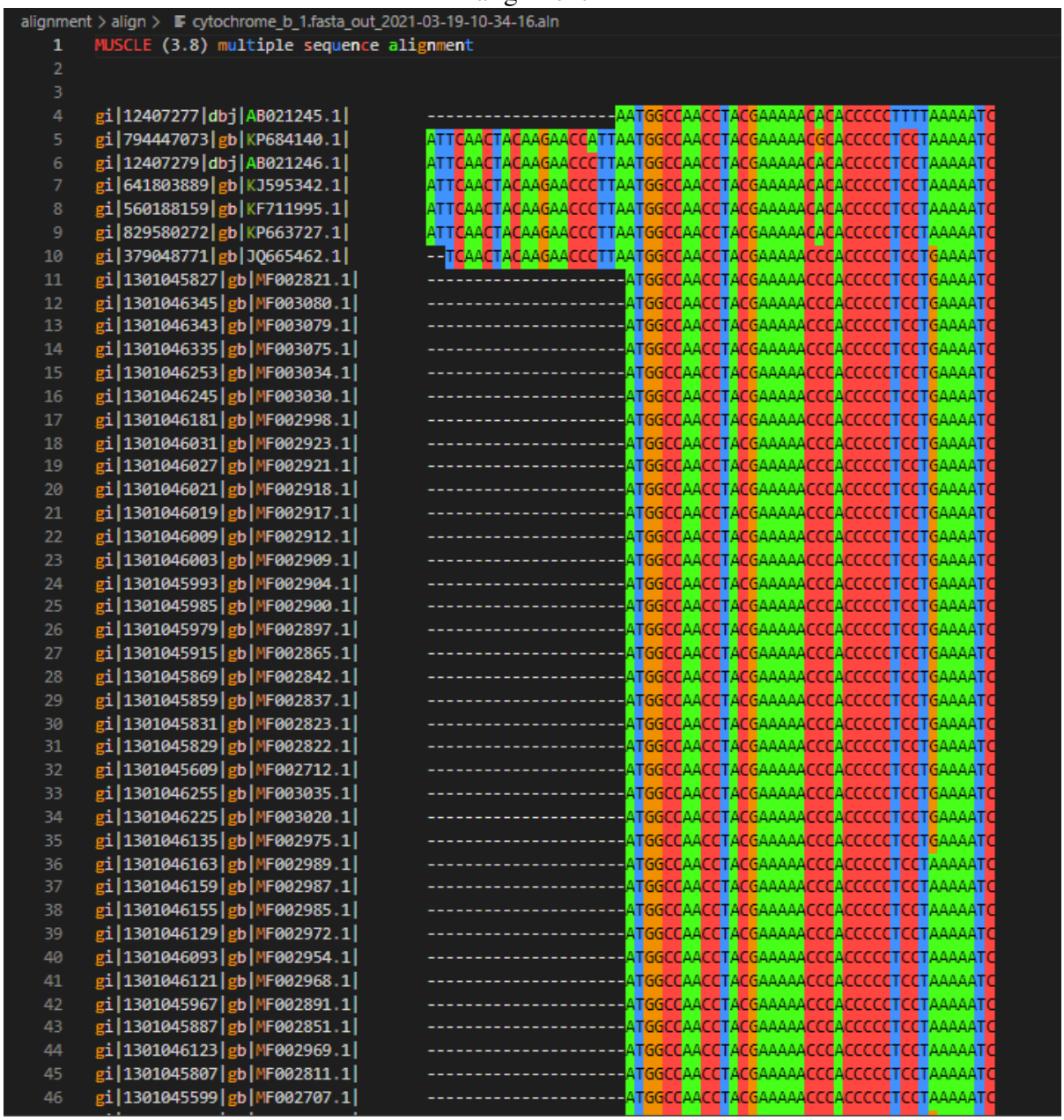
The *config* folder contains a *config.py* Python file where the software's GenBank API credentials are gathered and read from. The user will have to modify this file to input their own credentials for GenBank's API. The GenBank API is used to find homologous sequences to the input sequence via the BLAST algorithm and GenBank is one of the largest online databases of DNA sequences. Repliclade's implementation of the online BLAST algorithm is discussed in section 2.5.

The *alignment* folder in Repliclade's file system is where alignment related files are read from and stored. Under the *align* folder, there are output alignment files which are generated when Repliclade performs a multiple sequence alignment. The compiled executables used for executing multiple sequence alignments are contained in the *executables* folder. Repliclade uses these executables to perform these alignments which are integral to multiple components of Repliclade's overall functionality. In the executables folder, there are binaries compiled for both Windows OS and Mac/Linux OS. Depending on what the platform employed by the user, those relevant executables will automatically be used for the multiple sequence alignment of the sequences when running the program, making Repliclade a versatile cross-platform software.

The *sim* folder in the file system is where the main simulation file is housed, called *simulator.py*. This file controls the complete implementation of Repliclade. Contained within the sim folder is also the *results* folder. Here, every event from within the simulation is written to a JSON file for later reference by the user. As a result, the user is given the ability to go back through previous simulations and see the information in regards to every major event in those simulations.

The *util* folder houses all of the utility Python files. The *fileutil.py* file contains all of the Repliclade implementation which is integral to reading and writing to files in the file system. The *sequtil.py* file encompasses all functions relating to sequence modification and parameter extraction as well as multiple sequence alignment. This file is where the Coalescent process is also executed. Another file Repliclade uses is the *evoltree.py* file which houses all of the tree reconstruction methods used in the program.

Figure 2.2 An example of an alignment (.aln) file generated from a multiple sequence alignment



2.4 Biopython in Repliclade

One of the most popular Python libraries in regards to working with biological data is *Biopython*. Repliclade utilizes this library to perform many different biological processing-related tasks on DNA sequences. It provides different useful built-in functions to perform seemingly complicated computational tasks with respect to biology. Biopython also provides access to a very useful API which gives Repliclade access to GenBank. This is crucial in Repliclade since this is what the program uses to gather similar or homologous sequences in relation to the input sequence. Biopython is open-source and the repository for this library can be found at the following URL: `https://github.com/biopython/biopython`.

2.5 BLAST Algorithm

Once an input sequence has been specified in the DNA folder of the file system, Repliclade reads that sequence and executes an online query of GenBank using the BLAST algorithm. It uses a built in function that is a part of the Biopython library which provides an API connector for GenBank. In response, Repliclade retrieves 50 DNA sequences which are similar to the input sequence via the online query. This API response is saved as a file in XML format in the same DNA folder that is used to read input sequences into Repliclade.

The BLAST algorithm returns a list of HSP's (High Scoring Pairs). An HSP is a local alignment that achieves the highest score between the query and the return sequence. There are instances where there can be multiple HSP's for each hit sequence. Repliclade filters these out by taking only the highest scoring pair for each of these hit sequences which have multiple HSP's. These BLAST results are then used to perform an initial multiple sequence alignment on the group of related sequences which is later used to infer an ancestral sequence.

2.6 Multiple Sequence Alignment

After a successful BLAST response, Repliclade uses the returned DNA sequences to perform a multiple sequence alignment (MSA) of the resultant sequences. Repliclade utilizes an algorithm called MUSCLE for its MSA implementation. MUSCLE stands for MUltiple Sequence Comparison by Log-Expectation and is considered to be one of the best present-day MSA algorithms currently available. Repliclade previously implemented another algorithm called ClustalW2, which is also noteworthy. However, Repliclade now utilizes MUSCLE for multiple sequence alignment because it is considered more efficient and is known to produce better alignments overall. Repliclade takes the globally aligned DNA sequences to perform a multitude of operations to infer an ancestral DNA sequence as well as extract different sequence parameters vital to the ancestral sequence simulation.

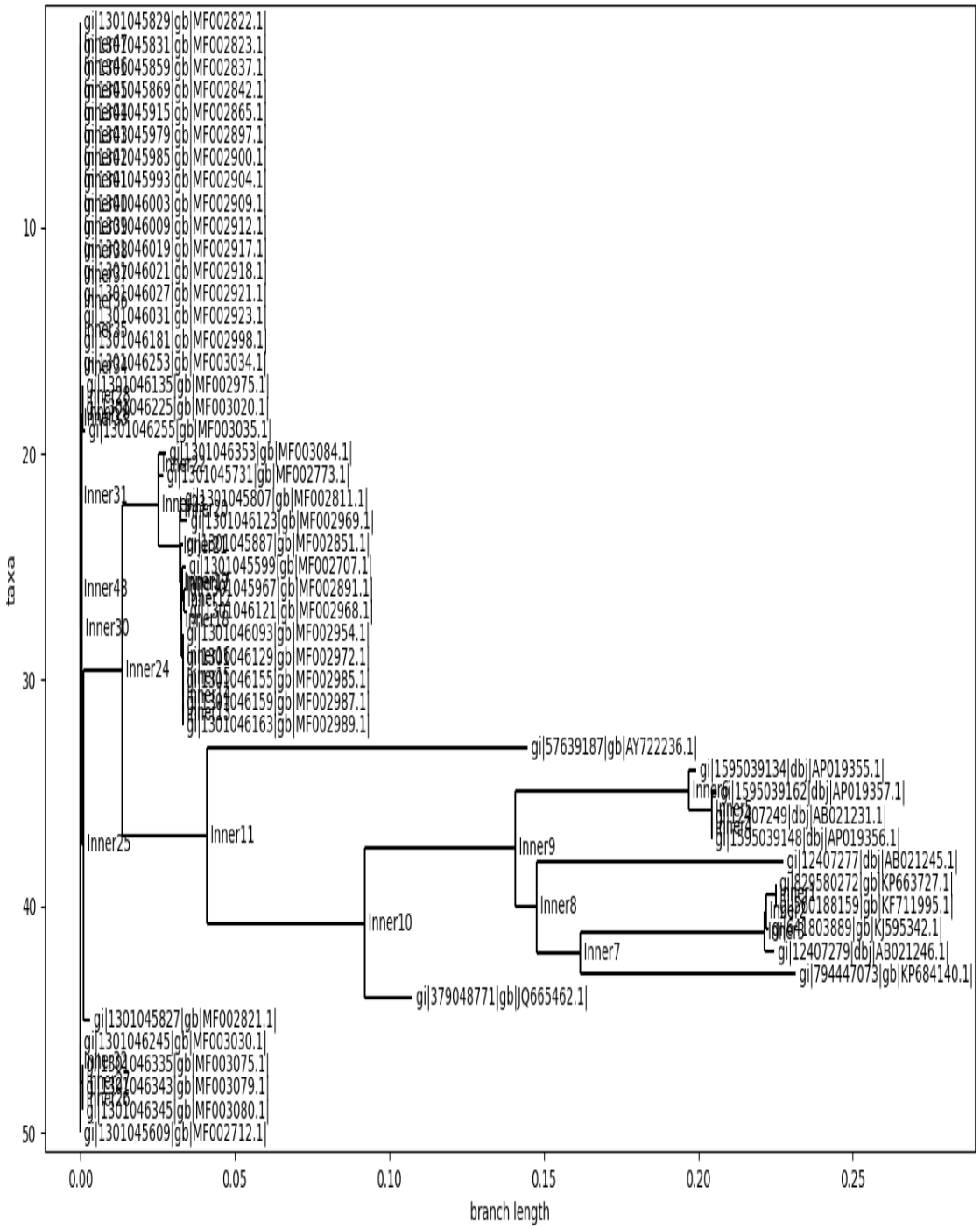
2.7 Calculating Conserved Regions in Repliclade

Repliclade uses the Shannon Entropy Score (Shannon, 1948) to calculate the entropy of specific regions in the DNA sequences after the initial multiple sequence alignment. A conserved region is a nucleotide or region in the DNA sequence that has remained unchanged or is much less likely to mutate as time progresses. Shannon's Entropy Score in Repliclade calculates the entropy of each nucleotide position in the aligned sequences. The following formula is used for each position in the alignment:

$$H(X) = -\sum_{i=1}^{n} P(x_i) log P(x_i) \tag{2.1}$$

Where X is a discrete random variable with outcomes $x_1, ..., x_n$ and $P(x_i)$ is the probability that x_i occurs. In this case, we have 5 possible outcomes. These encompass the 4 nucleotide base pairs A,G,T, and C as well as the gaps inserted into the alignments which are denoted as "-". Replclade produces a Python dictionary where each key is a number beginning at 0

Figure 2.3 An example of an alignment produced by MUSCLE displayed using the Biopython library



that denotes the starting position of the aligned sequences and each value is the entropy score calculated for that position. These conserved regions are later utilized by Repliclade during the forward simulation of an ancestral sequence.

2.8 Sequence Parameter Extraction

Once a set of aligned DNA sequences has been produced via MUSCLE alignment, Repliclade then performs an extraction of requisite values computationally which are useful in statistical population genetics. These values, such as effective population size (N_e) as described in Chapter 1, are integral to the forwards and backwards simulations that Repliclade performs. Repliclade first asks which method the user would like to use for the calculation of genetic variation (θ). It then calculates θ using the chosen method with a default mutation rate (μ) value. Subsequently, it also derives N_e since it now has values for θ and μ. After calculating θ and N_e, Repliclade will ask the user whether they would like to retain the default μ value or give their own value as an input. If the user inputs their own value, the genetic variation and effective population size will be calculated once again and Repliclade will provide new parameter values now that μ has changed. These values that have been acquired will then be used to infer an ancestral sequence to the group of aligned sequences.

Figure 2.4 An example output of θ and N_e using the Watterson Method in Repliclade

```
Please enter a choice of method for estimation of θ (variation)
You can choose either the Watterson Method (watterson)(Watterson 1975) or the Fay and Wu Method (faywu)(Fay and Wu 2000) watterson
Estimating effective population size using the Watterson method...
K:  31
alpha_n:  4.4792053383294235
theta_w:  6.920870480021763
Default μ:  1e-05
Effective Population size using Watterson estimator with default μ:  173021.76200054405
Coalescence time using Effective Population size from default μ:  678245.3070421326
Would you like to estimate the effective pop size with our own input? Please enter y or n:
```

2.9 The Coalescent in Repliclade

On a higher level, the implementation of the Coalescent in Repliclade follows the process from Chapter 1. It implements a pairwise merging of genealogies (or in this case, DNA sequences). This process repeats as time progresses until only one sequence is left. This becomes the inferred ancestral sequence. In this part of the program, effective population size is used to calculate the probability of a coalescent event occurring as shown in equation 1.29. When considering the forward evolution of a species, one naturally assumes mutations build up in that species through various means which ultimately culminate in a speciation event (a species diverging and becoming 2 separate species). If this is true for forward evolution, then a corresponding process should obtain when modeling movement backwards in time. Repliclade lets the user choose an evolutionary model from Chapter 1 to use in the Coalescent process. Once the user decides on their evolutionary model, Repliclade utilizes that model and evolves the sequences accordingly. Two sequences are chosen at random and a probability is calculated. If the conditions are met, the two sequences merge with each other and become one. If the conditions are not met, these DNA sequences are mutated via the chosen evolutionary model. Once the probability threshold of a Coalescent event is met, Repliclade executes a function which merges the 2 randomly chosen sequences into one. This direct ancestor to the 2 child genealogies is then used to further simulate the Coalescent process.

This process continues until only one DNA sequence remains. Since there is no way to determine which DNA sequence is older than the other, Repliclade must make a trade-off and infer an ancestral sequence through a randomized process like the Coalescent. This process can vary from being relatively fast to taking up to a couple of hours in some cases. Due to the randomized nature of the algorithm, this is something users must accept when simulating biological processes.

2.10 Simulation of Ancestral Sequence

Generating an ancestral sequence via the Coalescent process sets up Repliclade to implement one of its most important features. The program forges on to perform a forward simulation of the derived ancestral sequence. It will first prompt the user on which evolutionary model to use for the forwards simulation. Once the evolutionary model has been chosen, Repliclade then prompts the user on how many generations they would like to run the simulation across. The program also provides the option to calculate the amount of generations to run the simulation for by using the roughly estimated Coalescent time generated when calculating the values for N_e and θ. Whichever route the user chooses will ultimately initiate the simulation.

2.11 Implementation of Evolutionary Models

Evolutionary models available in Repliclade are contained in the *evolve.py* file in the *util* folder. Each evolutionary model has its own class structure for easy differentiation and object instantiation for the the computational aspect of the simulation. Every model class contains two class functions titled *evolve* and *calculate matrix*. The *evolve* function runs the sequence through the probability transition matrix for that particular model and determines whether a mutation occurs at each point in the sequence. Transition matrices in Repliclade are stored in the form of a Python dictionary. The key to this dictionary is the particular nucleotide base pair in question and the value is the array of mutational probabilities in the order of A, T, C, and G.

The function returns the sequence once it is complete. The *calculate matrix* function calculates the new probabilities of the transition matrix after each generation time interval. Since the evolutionary models are represented in Continuous Time Markov Chains, the re-calculation of the transition matrices are essential to generating the proper probabilities of mutation at any given interval of time.

Repliclade first checks the probability that a nucleotide base pair will stay the same and not mutate. If the probability is achieved, then the nucleotide will not mutate. If the probability threshold is not met, then Repliclade will perform the same process on the other nucleotide

Figure 2.5 The calculate matrix function for the K2P model. It is shown how Repliclade stores the values for the transition matrices

```
def calculate_matrix(self, alpha, beta, t):
    '''
    Markov model which defines the probability substitution matrix
    in the given unit of time
    '''
    transition = .25 + .25*(math.e**(-4*beta*t)) - .5*(math.e**(-2*(alpha + beta)*t))
    transversion = .25 - .25*(math.e**(-4*beta*t))
    same_nuc = 1 - transition - 2*transversion
    self.prb_matrix = {
        #     A    T    C    G
        'A': [same_nuc, transversion, transversion, transition],
        'T': [transversion, same_nuc, transition, transversion],
        'C': [transversion, transition, same_nuc, transversion],
        'G': [transition, transversion, transversion, same_nuc]
    }
```

Figure 2.6 The *evolve* class function utilizing the K2P transition matrix

```
def evolve(self, seq):
    '''
    Takes an input sequence and uses the Kimura model
    to evolve the sequence
    '''

    nuc_pos = {'A': 0, 'T': 1, 'C': 2, 'G': 3}

    ret_seq = ''
    for i in range(len(seq)):
        cur = seq[i]
        if cur == '-':
            ret_seq += cur
            continue
        first_roll = random.random()
        if first_roll <= self.prb_matrix[cur][nuc_pos[cur]]:
            ret_seq += cur
        else:
            for j in range(len(self.prb_matrix[cur])):
                if self.prb_matrix[cur][j] != 0:
                    roll = random.random()
                    if roll <= self.prb_matrix[cur][j] and self.prb_matrix[cur][j] != self.prb_matrix[cur][nuc_pos[cur]]:
                        cur = self.seq_list[j]
                        break
            ret_seq += cur
    self.t += 1
    self.calculate_matrix()
    return ret_seq
```

base pairs. This process repeats for each base pair on each sequence for each generation in the simulation. Figure 2.5 shows the K2P model implementation of the *evolve* function. We can see that after the process is complete for each sequence, the unit of time is incremented by 1 and the transition matrix is re-calculated for the next generation.

2.12 Forward Simulation of Ancestral Sequence

The evolutionary models are tracked via a model object array during the simulation. Each index in this array corresponds to the sequence index of the array of sequences which are being simulated. Originally, the simulation begins with one DNA sequence which is the ancestral sequence inferred by the Coalescent process. As time progresses, speciation or duplication events arise which provide new sequences to the simulator. New evolutionary model objects are created to be assigned to the new sequence as these duplication events arise.

Extinction events are also considered during the simulation. When an extinction event occurs, the sequence and corresponding model object are removed from the simulation. Extinction and Duplication events are tracked via a Python dictionary which are later written to a JSON format file in the *results* directory under the *sim* folder. The user can retroactively examine the duplication and extinction events of past simulations if needed.

The simulation runs for the user-specified amount of generations. During each generation, each sequence is run through its corresponding evolutionary model object which returns a mutated or non-mutated sequence. Probabilities for duplication, extinction, and indel events are then calculated. Currently, Repliclade is limited to only one of these events taking place per sequence per generation. If the probability thresholds for these events are met, then the corresponding event is executed and the simulation carries on. As expected, the run time for the simulation grows as the list of sequences being simulated increases in size.

Every DNA sequence that is generated in the simulation is also tracked within a simple general tree data structure. The child for a node in the tree is any sequence that has been duplicated from that parent node or parent sequence. Repliclade uses this structure to keep track of sequences and which other sequence they were duplicated from. This general tree is

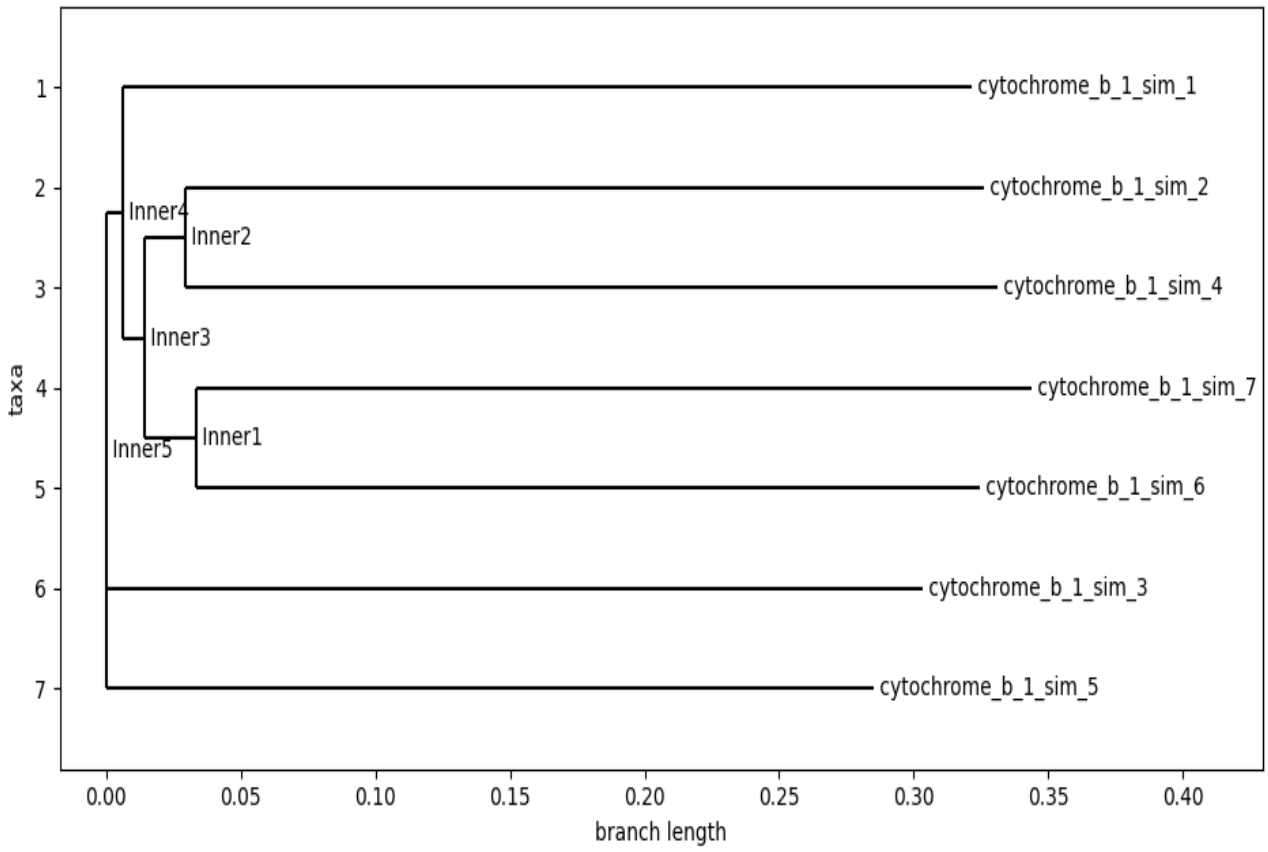
later displayed to the user post-simulation so they can envision the ”true phylogenetic tree” before Repliclade performs a phylogenetic reconstruction of the simulated sequences. Once the simulation completes, the simulator returns a list of every genealogy to exist within the simulation back to the main simulator function call.

2.13 Phylogenetic Reconstruction in Repliclade

Once the resultant sequences are returned from the simulator, Repliclade performs an alignment of the DNA sequences using MUSCLE before executing a phylogenetic reconstruction of the sequences. The program employs various methods for phylogenetic reconstruction. The methods used by Repliclade are the UPGMA, Neighbor-Joining, and Maximum Parsimony algorithms. The Biopython library provides utilities for performing phylogenetic reconstructions using each of these methods. The user is given the choice between these algorithms and Repliclade subsequently constructs a phylogenetic tree from the most recent alignment file.

This is the final step of the program, and now the user can determine which evolutionary model and method of phylogenetic reconstruction will work best for their input DNA sequence. Repliclade will then prompt the user whether they would like to run the program from the beginning once more. If yes, then Repliclade will start fresh and ask for an input sequence. All of the previous simulation results would still be present under the *results* folder.

Figure 2.7 An example of a phylogenetic reconstruction of the Cytochrome B gene after the ancestral sequence forward simulation using the Neighbor-Joining algorithm



Chapter 3

Results

3.1 Introduction

The Repliclade simulator evolves a set of sequences from an inferred ancestral sequence. The evolved sequences are analyzed phylogenetically and the resulting topology is compared to that of the ”true tree,” which was created and stored during the simulation. Topologies generated through analysis of the following three genes were evaluated in this manner:

1. Cytochrome B.
2. TNF
3. AGP

Four of the evolutionary models mentioned in Chapter 1 were employed for these simulations and subsequent analyses. These models were chosen because they are commonly used in analyses of molecular evolution and they are computationally efficient to implement.

3.2 Coalescent Results

The simulation of the Coalescent was performed on all 3 genes using the probability equations discussed in section 1.5. The evolutionary model used for the Coalescent was the same model used for each forward simulation to more faithfully mimic the same evolutionary processes. The ancestral sequence was inferred 100 times and the probability values were scaled by 100 to reduce the time required for Coalescence. The results are displayed in figure 3.1.

The averages were calculated for all Coalescent times pertaining to each evolutionary model with respect to every gene. The time to infer an ancestral sequence increases with various evolutionary models. The average Coalescent time for the HKY85 model on Cytochrome B in particular was 21 seconds. As was discussed in Chapter 1, implementing the HKY85 model entails considerably more calculations than the other models per iteration. It is expected that the average Coalescence time for the HKY85 model will rank higher than the other models. Coincidentally, the JC69 model ranked the lowest in terms of average Coalescence time. By following the same reasoning applied to the HKY85 model results, we see the reverse is applicable for the JC69 results as expected. The Watterson method was used for estimation of θ and N_e and the default μ value of $.00001$ was used for the Coalescent simulation.

3.3 Simulation of Evolution

The ancestral sequence derived via the Coalescent was then used to perform a forwards simulation of the specific gene. The simulation time was set to 50000 generations for this experiment. This number was chosen due to the average length of time for completion. The run-time was the most feasible for generating results in a reasonable amount of time. Only insert indel events were considered. Neither conserved regions or deletions were evaluated in this experiment. This was due to the ramifications on run-time for the simulation and was not feasible to generate the necessary results in a tractable amount of time if deletions and conserved regions were considered. Each gene was simulated 100 times for each of the 4 evolutionary models. In total, each gene was simulated 400 times for 50000 generations per simulation.

Figure 3.2 shows the average simulation times for each gene per evolutionary model. These results correlate well with the average Coalescent times. Gene duplication and extinction events can markedly impact the complexity of a simulation, leading to greater dispersion among run times. The average time is used to provide a better picture of the overall results. Here too, the computationally heavier evolutionary models such as HKY85 still averaged higher than the less computationally expensive models.

Table 3.1 Coalescent Results

Evolutionary Model	Average Coalescence Time (In Seconds)(100 Simulations)	Gene
K80	16	Cytochrome B
F81	16	Cytochrome B
JC69	12	Cytochrome B
HKY85	21	Cytochrome B
K80	18	TNF
F81	16	TNF
JC69	12	TNF
HKY85	22	TNF
K80	15	AGP
F81	14	AGP
JC69	10	AGP
HKY85	18	AGP

Table 3.2 Simulation Time Results

Evolutionary Model	Average Simulation Time (In Seconds)(100 Simulations)	Gene
K80	147	Cytochrome B
F81	121	Cytochrome B
JC69	108	Cytochrome B
HKY85	203	Cytochrome B
K80	160	TNF
F81	167	TNF
JC69	101	TNF
HKY85	217	TNF
K80	100	AGP
F81	89	AGP
JC69	79	AGP
HKY85	131	AGP

3.4 Phylogenetic Reconstruction

For the experiment, each of the 4 phylogenetic reconstruction methods was used for each phylogeny generated by the simulation. After each simulation, the resultant DNA sequences were aligned using MUSCLE and then subsequently rebuilt using selected tree reconstruction methods. Using the Biopython library, we implemented the UPGMA, NJ, and MP tree reconstruction functions to infer a phylogeny for the evolved sequences. For Maximum Likelihood reconstruction, we also used the Biopython wrapper for the PhyML executable file to reconstruct the phylogeny.

Each of the tree reconstructions as well as the ”true” phylogenetic tree were saved to a file in the Repliclade file system under the *trees* directory in the *results* folder. These files were later read and processed by a Python utility file developed to quantify topological disparities between inferred phylogenies and the ”true” tree.

3.5 Phylogenetic Reconstruction Similarity Scores

A Python utility file titled *replicladeinsights.py* was developed to process the files and phylogenies generated by the simulation. The Python file generates a similarity score from each tree reconstruction in regards to the true tree. The similarity score comes from a myriad of different attributes in relation to the phylogenetic trees. The characteristics evaluated are:

1. Terminal nodes between the trees (leaf nodes)
2. Non-Terminal nodes between the trees (non-leaf nodes)
3. Pre-Terminal nodes between the trees (parent nodes of terminal nodes)
4. Interior Branch Test

The scores for each evolutionary model are calculated by summing the scores for each individual reconstruction when compared to the true tree using a particular tree building algorithm. Similarity scores are then assigned to each phylogenetic tree reconstruction method as well as

to each evolutionary model. This enables Repliclade to evaluate performance characteristics of each model and phylogenetic inference algorithm for individual data matrices. The attributes used for similarity scoring are minimal, though adequate, to assess the overall characteristics of the true tree as well as the reconstructed trees.

Figures 3.3, 3.4, 3.5, and 3.6 display the similarity scores for each evolutionary model and tree building algorithm, starting with the JC69 model. These results indicate that the Maximum Likelihood algorithm yielded the highest similarity score to the true tree. Performance characteristics of the UPGMA and Maximum Parsimony algorithms generally placed them second and third, but the order of the two varied. Across all of the experiments, the Neighbor Joining algorithm most commonly had the lowest total similarity score. However, in the case of the TNF gene, the Neighbor Joining algorithm garnered a higher similarity score than the UPGMA algorithm when the JC69 evolutionary model was employed.

Table 3.3 JC69 Similarity scores

Gene	Tree Reconstruction Method	Similarity Score to True Tree
Cytochrome B	NJ	151.55
Cytochrome B	UPGMA	157.03
Cytochrome B	Maximum Parsimony	156.75
Cytochrome B	Maximum Likelihood	164.37
TNF	NJ	172.48
TNF	UPGMA	171.33
TNF	Maximum Parsimony	177.12
TNF	Maximum Likelihood	178.59
AGP	NJ	144.99
AGP	UPGMA	142.76
AGP	Maximum Parsimony	146.08
AGP	Maximum Likelihood	149.75

Table 3.4 K80 Similarity scores

Gene	Tree Reconstruction Method	Similarity Score to True Tree
Cytochrome B	NJ	143.12
Cytochrome B	UPGMA	142.78
Cytochrome B	Maximum Parsimony	149.19
Cytochrome B	Maximum Likelihood	155.97
TNF	NJ	142.96
TNF	UPGMA	145.64
TNF	Maximum Parsimony	150.59
TNF	Maximum Likelihood	156.83
AGP	NJ	161.66
AGP	UPGMA	166.85
AGP	Maximum Parsimony	168.69
AGP	Maximum Likelihood	168.67

Table 3.5 F81 Similarity scores

Gene	Tree Reconstruction Method	Similarity Score to True Tree
Cytochrome B	NJ	156.42
Cytochrome B	UPGMA	168.58
Cytochrome B	Maximum Parsimony	162.85
Cytochrome B	Maximum Likelihood	167.17
TNF	NJ	157.23
TNF	UPGMA	159.84
TNF	Maximum Parsimony	161.27
TNF	Maximum Likelihood	163.26
AGP	NJ	153.99
AGP	UPGMA	163.56
AGP	Maximum Parsimony	161.37
AGP	Maximum Likelihood	163.42

Table 3.6 HKY85 Similarity scores

Gene	Tree Reconstruction Method	Similarity Score to True Tree
Cytochrome B	NJ	164.87
Cytochrome B	UPGMA	173.83
Cytochrome B	Maximum Parsimony	170.73
Cytochrome B	Maximum Likelihood	172.68
TNF	NJ	170
TNF	UPGMA	178.5
TNF	Maximum Parsimony	174.94
TNF	Maximum Likelihood	178.32
AGP	NJ	170.96
AGP	UPGMA	177.49
AGP	Maximum Parsimony	174.06
AGP	Maximum Likelihood	177.03

Chapter 4

Discussion

4.1 Coalescent

Repliclade has varying resultant times for its operations depending on which evolutionary models and phylogenetic reconstruction algorithms are used. The computational efficiency of certain evolutionary models have an adverse effect on performance when simulating an ancestral sequence. This is particularly evident when simulating the Coalescent using the HKY85 model in the experiments. This is due to the fact that HKY85 not only considers base frequencies, but it also considers transitions and transversions in its calculations (Hasegawa et al., 1985). The run-time for a randomized algorithm such as the Coalescent is dependent on random variables as well as the computational efficiency of the evolutionary models used. The results for the set of experiments presented herein were scaled due to time constraints. Assuming non-scaled results, we can hypothesize the Coalescent algorithm also scaling per the results in run-time.

4.2 Simulation Results

The simulation results were also a heavily randomized process. This is due to the randomness of extinction and duplication events which occurred within the simulation as well as indel events, which did not effect the run-time as adversely. As evolution progresses in nature, certain species or genealogies can go extinct due to many different evolutionary or environmental forces. In contrast, species can diverge into separate genealogies via duplication events. Repliclade incorporates these events and therefore the resultant run-times of the simulations can vary

widely. To remedy the differing results, we have taken the averages of the execution times to better assess disparities in simulation time. The results indicate that evolutionary models with an increased number of calculations per iteration tend to increase the run-time of the simulation. This is in line with previous studies of the performance of evolutionary models in Monte Carlo simulations (Brown et al., 2018). Alternatively, evolutionary models such as the JC69 model (Jukes et al., 1969) yielded a lower run-time due to the lower amount of calculations required for each iteration.

For this experiment, conserved regions and deletions were not considered due to time constraints and performance implications that these operations confer on the simulation. Accordingly, these analyses focused on micro-indels only, which have an insert size between 1 and 50 base pairs. This range was chosen because micro-indels are theorized to be common contributors to cancers in the human germline (Gonzales et al., 2007) and may be utilized to explore the formation of deleterious mutations in tumor-suppressor genes and proto-oncogenes. Additionally, micro-indels had a negligible effect on the performance of the simulation. These characteristics made micro-indels especially attractive to incorporate in these simulations.

4.3 Phylogenetic Tree Reconstruction

The metrics used to evaluate phylogenetic tree reconstruction of the evolved sequences from the simulation were listed in section 3.5. There are many ways to evaluate the fidelity of phylogenetic reconstruction. A method that the Repliclade utility file uses when evaluating the derived phylogenies constructed by the Maximum Likelihood method is the Interior Branch Test (Felsenstein, 1981). This test identifies interior branches in a topology that have a length that is significantly greater than 0. Interior branches that have a length that is not significantly greater than 0, suggest that alternate patterns for branching in that particular region of the tree were not ruled out by the Maximum Likelihood algorithm.

Terminal and non-terminal nodes are essential to the phylogenetic tree scoring process in Repliclade as well. The Terminal nodes are compared and intersected between the true tree and the reconstructed tree being evaluated. The ratio of intersection terminal nodes to total unique

nodes between the trees (these will be identical) is used as part of the overall scoring of the reconstructed tree. By Corollary, non-terminal nodes are evaluated in the same manner. The ratio of the intersection of non-terminal nodes to the true tree is also used as part of the total scoring mechanism. These particular scoring mechanisms take a more parsimonious approach, utilizing differences between the two trees to generate a score for that particular tree building algorithm.

4.4 Evaluating the Results

4.4.1 JC69 Tree Reconstruction

As was discussed in chapter 3, the simulation of the ancestral DNA sequence using the JC69 model was more efficient in terms of run-time and computational complexity. Accordingly, one could assume the resultant sequences derived from the ancestral sequence simulation would have a lower tree reconstruction similarity score. While this was true for most genes in our study, an exception was the TNF gene. Figure 3.3 reveals that the TNF gene had a similarity score of over 170 for each of the phylogenetic tree reconstruction algorithms after being evolved using the JC69 model. As such, TNF outperformed the other genes substantially when assessing the tree building algorithms. This further strengthens the case for utilizing the JC69 model when building phylogenetic trees of the TNF gene as has been done in previous studies (Mathew et al., 2013). The AGP gene, on the other hand, had much lower similarity scores for all 4 of the tree building algorithms when compared to the other 2 genes in our study, suggesting that the constraints on its evolution may not be well reflected in the models evaluated here.

4.4.2 K80 Tree Reconstruction

The K80 model introduced in Chapter 1 discriminates between the instantaneous rates of transitions and transversions (Kimura, 1980). This more closely resembles evolutionary forces that are operative in the diversification of a vast majority of previously studies gene families (Norris et al, 2015). Due to this characteristic, we would expect to see an improvement in tree reconstruction similarity scores over the JC69 model due to the trade-off in efficiency that the K80 model makes. It is therefore somewhat surprising that the K80 model yielded topologies with poorer performance metrics in the cases of both the Cytochrome B and TNF genes in analyses of our simulation results. An exception to this observation is the AGP gene, which scored significantly higher in all tree reconstruction algorithm categories using the K80 model than what was observed in the JC69 model. These results underscore the importance of evaluating model performance for a locus of interest before engaging in computationally intensive phylogenetic analysis (Collins et al., 2012). Conceivably, the AGP gene may have transition and transversion rates that more closely adhere to those anticipated under K80 than the other genes in our study. If so, the distance calculations employed during phylogenetic reconstruction may have more closely resembled that bias.

4.4.3 F81 Tree Reconstruction

The F81 evolutionary model incorporates base frequencies in its calculations and was the first to do so (Felsenstein, 1981). Due to this characteristic, base frequencies of the group of original sequences pre-coalescent needed to be calculated to use as parameters for the simulation. The phylogenetic tree reconstruction similarity scores generated by our experiment show consistency across most of the tree building methods for each gene. This result could partially be attributed to base frequencies and their effects on evolutionary change and nucleotide base mutations. Previous studies have proposed that there is appreciable interplay between base frequencies and the dynamics of selection, drift, and mutation at the population level (Knudsen et al., 2005). The F81 model produced the highest similarity score for the Cytochrome B gene when used in conjunction with the UPGMA algorithm, which deviates from the general

pattern that the Maximum Likelihood algorithm usually outperforms the distance based algorithms (Whelan et al., 2001). This may be, in part, be related to the unique cellular location of Cytochrome B which resides in Mitochondrial genome and is known to evolve more rapidly (Brown et al., 1982). By corollary, distance based algorithms may be more efficient, in general, when building phylogenies from loci in the Mitochondrial genome.

4.4.4 HKY85 Tree Reconstruction

The HKY85 model incorporates base frequencies as well as transition and transversion ratios in its calculations (Hasegawa et al., 1985). Similar to the F81 model's Repliclade implementation, base frequencies derived from the group of original homologous pre-coalescent sequences were used as parameters in the evolutionary model. The phylogenetic tree reconstruction similarity scores for the HKY85 model show an average higher score than the other evolutionary models for every tree building method. This aligns with results from previous studies, which show that the HKY85 tends to be more accurate due to the powerful combination of strengths of both the K80 and F81 evolutionary models (Yang, 1993). Table 3.6 also shows the UPGMA tree building method scoring highest in the all of the genes when used in conjunction with HKY85. The UPGMA reconstruction method had a higher similarity score than the other methods. This is similar to the F81 model similarity scores, where the UPGMA algorithm scored the highest similarity score for the Cytochrome B gene. For each gene, the maximum likelihood method came extremely close to the similarity score associated with the UPGMA algorithm. These results indicate a randomized bias in the similarity scores and simulations. This observation is consistent with previous studies, which show that base frequency consideration and randomized processes in a group of sequences can obscure phylogenetic reconstruction accuracy (Davalos et al., 2008).

Chapter 5

Conclusion

Repliclade provides a novel resource for biological researchers that enables them to assess which methods are best suited to the unique properties of the gene selected for phylogenetic analysis and the biological lineage under study. The program supports quantitative comparison of multiple evolutionary models as well as a multitude of phylogenetic inference methods. Repliclade incorporates well-vetted methodologies such as the Coalescent process and methods of sequence parameter extraction. Collectively, these methods are utilized to infer the composition of an ancestral sequence and subsequently simulate diversification of this locus within a biological lineage. Repliclade incorporates a separate utility file to analyze reconstructed phylogenies and score them based on accuracy relative to the true phylogenetic tree of the simulation. Some tree building algorithms tend to be more accurate for certain genes or gene families. These results underscore the importance of informed model selection prior to computationally intensive phylogenetic analyses.

5.1 Future Work

In the future, Repliclade could incorporate more tree building algorithms and evolutionary models. In particular, Repliclade could add the GTR (General Time Reversible) model to its arsenal. Future analyses will explore the interplay between a broader range of candidate loci and evolutionary model combinations. Deletions and conserved regions could be considered for future simulations and the results could be tested with all of the tree reconstruction methods. Repliclade could also incorporate other methods of tree similarity scoring when ascertaining

the similarity of the reconstructed phylogenetic trees. There are also many more methods of sequence parameter extraction for various values, and these could also be incorporated into the program to provide an expanded range of model and algorithmic choices to the user.

LIST OF REFERENCES

[1] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.

[2] Jeremy M. Brown and Robert C. Thomson. Evaluating model performance in evolutionary biology. *Annual Review of Ecology, Evolution, and Systematics*, 49:95–114, 2018.

[3] John A. Capra and Mona Singh. Predicting functionally important residues from sequence conservation. *Bioinformatics*, 23:1875–1882, 2007.

[4] Reed A. Cartwright. Problems and solutions for estimating indel rates and length distributions. *Molecular Biology and Evolution*, 26:473–480, 2009.

[5] Jose Castresana. Cytochrome b phylogeny and the taxonomy of great apes and mammals. *Molecular Biology and Evolution*, 18:465–471, 2001.

[6] Brian Charlesworth. Effective population size and patterns of molecular evolution and variation. *Genetics*, 10:195–205, 2009.

[7] Qi-Long Chen, Xin-Sheng Tang, Wen-Juan Yao, and Shun-Qing Lu. Bioinformatics analysis the complete sequences of cytochrome b of takydromus sylvaticus and modeling the tertiary structure of encoded protein. *International Journal of Biological Sciences*, 6:596–602, 2009.

[8] Benny Chor and Tamir Tuller. Maximum likelihood of evolutionary trees: hardness and approximation. *Bioinformatics*, 21:97–106, 2005.

[9] Rupert A. Collins, Laura M. Boykin, Robert H. Cruickshank, and Karen F. Armstrong. Barcoding's next top model: an evaluation of nucleotide substitution models for specimen identification. *Methods in Ecology and Evolution*, 3:457–465, 2012.

[10] Liliana Davalos and Susan L. Perkins. Saturation and base composition bias explain phylogenomic conflict in plasmodium. *Genomics*, 91:433–442, 2008.

[11] Mauro Delgi Esposti, Simon De Vries, Massimo Crimi, Anna Ghelli, Tomaso Patarnello, and Axel Meyer. Michondrial cytrochrome b: evolution and structure of the protein. *Biochimica et Biophysica*, 1143:243–271, 1993.

[12] Justin C. Fay and Chung-I Wu. Hitchhiking under positive darwinian selection. *Genetics*, 155:1405–1413, 2000.

[13] Joseph Felsenstein. Estimating effective population size from samples of sequences: inefficiency of pairwise and segregating sites as compared to phylogenetic estimates. *Genetics Research*, 59:139–147, 1991.

[14] Yun-Xin Fu. Estimating mutation rate and generation time from longitudinal samples of dna sequences. *Society for Molecular Biology and Evolution*, 18:620–626, 2001.

[15] Courtney E. Gonzales, Paul Roberts, and Marc Ostermeier. Fitness effects of single amino acid insertions and deletions in tem-1 -lactamase. *Journal of Molecular Biology*, 431:2320–2330, 2019.

[16] Kelly D. Gonzales, Kathleen A. Hill, Kai Li, Wenyan Li, William A. Scaringe, Ji-Cheng Wang, Dongqing Gu, and Steve S. Sommer. Somatic microindels: Analysis in mouse soma and comparison with the human germline. *Human Mutation*, 28:69–80, 2007.

[17] Matthew W. Hahn. *Molecular Population Genetics*, volume I. Oxford University Press, New York, 2018.

[18] Masami Hasegawa, Hirohisa Kishino, and Taka aki Yano. Dating of the human-apesplitting by a molecular clock of mitochondrial dna. *Journal of Molecular Evolution*, 22:160–174, 1985.

[19] Ian H. Holmes. Solving the master equation for indels. *BMC Bioinformatics*, 18:25, 2017.

[20] Motoo Kimura. Estimation of evolutionary distances between homologous nucleotide sequences. *Professional National Academy of Sciences*, 78:454–458, 1981.

[21] Bjarne Knudsen and Michael M. Miyamoto. Using equilibrium frequencies in models of sequence evolution. *BMC Evolutionary Biology*, 5:21:1471–2148, 2005.

[22] Mary K. Kuhner, Jon Yamato, and Joseph Felsenstein. Estimating effective population size and mutation rate from sequence data using metropolis-hastings sampling. *Genetics*, 140:1421–1430, 2013.

[23] Pietro Lio and Nick Goldman. Models of molecular evolution and phylogeny. *Genome Research*, 8:1233–1244, 1998.

[24] Michael Lynch. Rate, molecular spectrum, and consequences of human mutation. *PNAS*, 107:961–968, 2009.

[25] Michael Lynch. Evolution of the mutation rate. *Trends in Genetics*, 26:345–352, 2010.

[26] Marina Mathew, Kenneth W. Beagley, Peter Timms, and Adam Polkinghome. Preliminary characterisation of tumor necrosis factor alpha and interleukin-10 responses to chlamydia pecorum infection in the koala (phascolarctos cinereus. *PLoS ONE*, 8:e59958, 2013.

[27] Tomotoka Matsumoto, Hiroshi Akashi, and Ziheng Yang. Evaluation of ancestral sequence reconstruction methods to infer nonstationary patterns of nucleotide substitution. *Genetics*, 200:873–890, 2015.

[28] Laura May-Collado and Ingi Agnarrson. Cytochrome b and bayesian inference of whale phylogeny. *Molecular Phylogenetics and Evolution*, 38:344–354, 2006.

[29] Julienne M. Mullaney, Ryan E. Mills, W. Stephen Pittard, and Scott E. Devine. Small insertions and deletions (indels) in human genomes. *Human Molecular Genetics*, 19:131–136, 2010.

[30] Ulberto Pozzoli, Giorgia Menozzi, Matteo Fumagalli, Matteo Cereda, Giacomo P Comi, Rachele Cagliani, Nereo Bresolin, and Manuela Sironi. Both selective and neutral processes drive gc content evolution in the human genome. *BMC Evolutionary Biology*, 8:99:1471–2148, 2008.

[31] Miguel Rocha and Pedro G. Ferreira. *Bioinformatics Algorithms*, volume I. Academic Press, Oxford, UK, 2018.

[32] Naruya Saitou and Masatoshi Nei. The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Molecular Biological Evolution*, 4:406–425, 1987.

[33] Rafael Sanjuan, Miguel R. Nebot, Nicola Chirico, Louis M. Mansky, and Robert Belshaw. Viral mutation rates. *Journal of Virology*, 84:9733–9748, 2010.

[34] Armin O. Schmitt and Hanspeter Herzel. Estimating the entropy of dna sequences. *MPI fur molekulare Genetik*, 1888:369–377, 1997.

[35] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 1948.

[36] Sara Sheehan, Kelley Harris, and Yun S. Song. Estimating variable effective population sizes from multiple genomes: A sequentially markov conditional sampling distribution approach. *Genetics*, 194:647–662, 2013.

[37] Robert R. Sokal and Charles D. Michener. A statistical method for evaluating systematic relationships. *University of Kansas Science Bulletin*, 38:1409–1438, 1958.

[38] Arlin Stoltzfuz and Ryan W. Norris. On the causes of evolutionary transition:transversion bias. *Society for Molecular Biology and Evolution*, 2015.

[39] Way Sung, Matthew S. Ackerman, Marcus M. Dillon, Thomas G. Platt, Clay Fuqua, Vaughn S. Cooper, and Michael Lynch. Evolution of the insertion-deletion mutation rate across the tree of life. *G3*, 6:2583–2591, 2016.

[40] Fujima Tajima. Infinite-allele model and infinite-site model in population genetics. *Journal of Genetics*, 75:27–31, 1996.

[41] Fujima Tajima. Infinite-allele model and infinite-site model in population genetics. *Journal of Genetics*, 75:27–31, 1996.

[42] Eric Talevich, Brandom M. Invergo, Peter JA Cock, and Brad A. Chapman. Bio.phylo: A unified toolkit for processing, analyzing and visualizing phylogenetic trees in biopython. *BMC Bioinformatics*, 13:209:1471–2105, 2012.

[43] Jeffrey L. Thorne, Hirohisa Kishino, and Joseph Felsenstein. An evolutionary model for maximum likelihood alignment of dna sequences. *Journal of Molecular Evolution*, 33:114–124, 1991.

[44] Jr. W. Robert Fleischmann. *Medical Microbiology*, volume IV. Oxford University Press, Galveston, TX, 2018.

[45] John Wakeley and Tsuyoshi Takahashi. Gene genealogies when the sample size exceeds the effective size of the population. *Molecular Biology and Evolution*, 20:208–213, 2003.

[46] J Wang, E Santiago, and A Caballero. Prediction and estimation of effective population size. *Heredity*, 117:193–206, 2016.

[47] Jinliang Wang. Estimation of effective population sizes from data on genetic markers. *Philosophical Transactions of the Royal Society B*, 360:1395–1409, 2005.

[48] Simon Whelan and Nick Goldman. A general empirical model of protein evolution derived from multiple protein families using a maximum-likelihood approach. *Molecular Biology and Evolution*, 18:691–699, 2001.

[49] Sewall Wright. Evolution in mendelian populations. *Genetics*, 16:97–193, 1930.

[50] Ziheng Yang. Estimating the pattern of nucleotide substitution. *Journal of Molecular Evolution*, 39:105–111, 1994.

[51] Ziheng Yang. *Molecular Evolution: A Statistical Approach*. Oxford University Press, New York, NY, 2014.

[52] Ziheng Yang and Bruce Rannala. Molecular phylogenetics: principles and practice. *Genetics*, 13:303–314, 2012.

[53] Ziheng Yang and Anne D. Yoder. Estimation of the transition/transversion rate bias and species sampling. *Journal of Molecular Evolution*, 48:274–283, 1999.

[54] Isao Yuasa, Hiroaki Nakamura, Lotte Henke, Jurgen Henke, Mayumi Nakagawa, Yoshito Irizawa, and Kazuo Umetsu. Characterization of genomic rearrangements of the 1-acid glycoprotein/ orosomucoid gene in ghanaians. *Journal of Human Genetics*, 46:572–578, 2001.

Appendix A: Replclade Code

The repository for Repliclade can be found on Github at the following url:
`https://github.com/hakidehari/RepliClade`