Efficient Parameter Estimation for Human Microsatellite Mutation

Glenn Galvizo, under Dr. Floyd Reed

December 11, 2018

Abstract

Variation in DNA sequences allow us to describe the history of humanity. A microsatellite is a form of genetic variation where short DNA sequences are repeated in tandem. Each microsatellite variant differs in how many times the short sequence is repeated. Human history inferences can be made by surveying how the number of repeats change across generations. Given a mutation and evolution model for human microsatellites, the question this project aims to answer is, "What are the most likely parameters for this model?".

To find the best parameters given a set of observed microsatellite samples is to maximize a likelihood. For this problem though, the maximum cannot be found numerically in a reasonable amount of time. My methods to circumvent this were as follows: (1) Observations were collected from an existing database. (2) Simulated samples were generated using a coalescent approach, which avoids working on individuals that do not directly contribute to a sample's evolutionary history. (3) To compare an observed sample and a simulated sample, the angular distance was used. (4) An approximate likelihood calculation was used in lieu of the exact likelihood to account for the low frequency of exact observation – simulation matches. (5) A Markov Chain Monte Carlo (MCMC) method was used to produce samples from the approximate likelihood.

Preliminary results show strong evidence for the existence of optimal parameters for our microsatellite mutation model. With defined parameters, future work entails using the same methodology for parameter determination of more complex demographic models.

Keywords: Markov Chain Monte Carlo, Approximate Bayesian Computation, Backward Simulation, Human Microsatellite DNA, Microsatellite Mutation Model

Contents

Chapter 1

Introduction

Around $200,000$ years ago, modern humans evolved in Africa. $120,000 - 160,000$ years after this, humans migrated out of Africa into Eurasia [3]. In addition to this, we know that modern humans have interbred with other *Homo sapiens*: Neanderthals and Denisovans [21]. The main questions left to answer about the evolutionary history of modern humans are: "How big was each population at time t?" "Can we get a more accurate date for the migration out of Africa?" "Are there more species we have admixture with?" And the overarching question: "What is the complete demographic model for modern humans?"

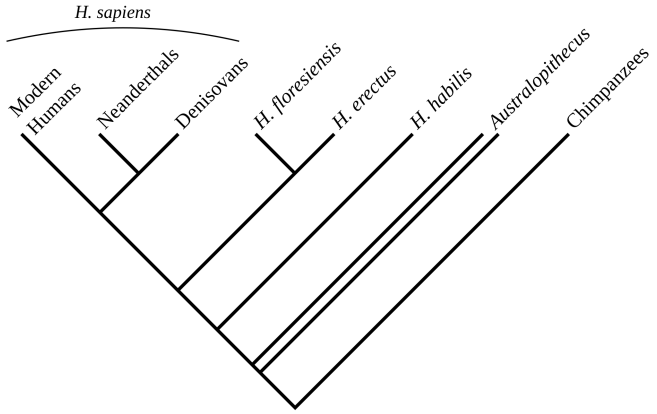

Figure 1.1: Phylogeny tree of humans and related species [18].

Where did we get our current estimates? The answer (for the majority) is genetic variation. We can infer evolutionary history by tracking how DNA sequences vary (or mutate) from generation to generation. This is done by constructing different demographic models, and determining the likelihood of said models. This paper addresses the first step toward being able to do so: finding a reasonable evolutionary model for a *single* population.

The rest of this paper is organized as follows: In chapter 2, we describe the two most common measures of genetic variation. In chapter 3, we describe how to efficiently simulate a microsatellite sample from a coalescing population. In chapter 4, we discuss how we model microsatellite mutation. In chapter 5, we briefly describe our observed data. In chapter 6, we discuss our methods for maximum likelihood estimation. In chapter 7, we describe our methodology and results. In chapter 8, we discuss what we plan to do after this project.

Chapter 2

Single Nucleotide Polymorphisms vs. Microsatellites

In this chapter we briefly cover what polymorphisms, single nucleotide polymorphisms, and microsatellites are.

2.1 Polymorphism

Polymorphism is the existence of two or more *morphs* (forms) in some population. These variations are generated in humans by the mutational process, recombination, and mating. In terms of genetics, polymorphisms describe different alleles within a gene. These changes may occur in both coding regions (i.e. a polymorphism results in different protein being produced) and non-coding regions. Once this change occurs in some individual, all the offspring of said individual will possess this new morph until another mutation occurs. These changes may also select toward certain alleles, or solely exist as a result of genetic drift. In most cases though, polymorphisms are not functional and are selectively neutral [25]. Consequently this paper does not review the functional aspect of certain morphs and assumes this selective neutrality.

2.2 Single Nucleotide Polymorphisms

Single nucleotide polymorphisms (abbreviated as SNPs), are single nucleotide differences in a DNA sequence. The set of nucleotides that compose such a sequence are $\{\texttt{A}, \texttt{T}, \texttt{C}, \texttt{G}\}$, which represent adenine, thymine, cytosine and guanine respectively. To be considered a SNP and not a point mutation, the least frequent allele must be present in at least 1% of a population. An example of an SNP with 2 variants is given by the set S below. The differences in this SNP are the `G` and `A` in bold.

$$
\begin{aligned}
S &= \{S_1, S_2\} \\
S_1 &= \ldots \texttt{GGGTCACAT}\mathbf{\texttt{G}}\texttt{GTTACAGTA} \ldots \\
S_2 &= \ldots \texttt{GGGTCACAT}\mathbf{\texttt{A}}\texttt{GTTACAGTA} \ldots
\end{aligned}
$$

Theoretically a SNP could have up to 4 variants, but most are biallelic in nature. Given that two states exist for most SNPs, multiple SNPs are typically used to determine one's evolutionary history [14].

2.3 Microsatellites

Microsatellites are sequences in DNA that are repeated in tandem. These sequences range from 1 to 5 nucleotides long, with copies of this sequence (measured in *repeat units*) ranging from ~ 3 nucleotides to 100 [19, 8]. An example of a microsatellite variation with the AT sequence and 5 repeat units in arbitrary sequence of DNA is given below. The microsatellite variation itself is represented in bold.

$$\ldots \texttt{AACG}\mathbf{\texttt{ATATATATAT}}\texttt{GGCTA}\ldots$$

In more precise terms, a microsatellite is a set of nucleotide sequences whose elements are only varied by repeat units. The microsatellite itself belongs to a class of repeat polymorphisms known as short tandem repeat polymorphisms (STRP). An example of a repeat polymorphism with the repeat sequence CA is given by the set M below.

$$\begin{aligned}
M &= \{M_1, M_2, M_3, M_4, M_5, M_6, M_7\} \\
M_1 &= \ldots \texttt{GGGT}\mathbf{\texttt{CACACACACACACACACA}}\texttt{GTTAC}\ldots \\
M_2 &= \ldots \texttt{GGGT}\mathbf{\texttt{CACACACACACACACA}}\texttt{GTTACAG}\ldots \\
M_3 &= \ldots \texttt{GGGT}\mathbf{\texttt{CACACACACACACACA}}\texttt{GTTACAGTA}\ldots \\
M_4 &= \ldots \texttt{GGGT}\mathbf{\texttt{CACACACACACA}}\texttt{GTTACAGTAAA}\ldots \\
M_5 &= \ldots \texttt{GGGT}\mathbf{\texttt{CACACACACA}}\texttt{GTTACAGTAAATA}\ldots \\
M_6 &= \ldots \texttt{GGGT}\mathbf{\texttt{CACACACA}}\texttt{GTTACAGTAAATAGG}\ldots \\
M_7 &= \ldots \texttt{GGGT}\mathbf{\texttt{CACACA}}\texttt{GTTACAGTAAATAGGAA}\ldots
\end{aligned}$$

Microsatellites are of evolutionary interest due to their repeat unit instability across generations. In the 1990s, microsatellites and other repeat polymorphisms were replaced by nucleotide polymorphisms due to SNP's abundance in the genome, their recoverability, and the resulting functional consequences if the sequence is in a coding region [11, 2]. When compared to SNPs though, microsatellites mutate at rates between 10^{-3} to 10^{-6} mutations per generation, 1 to 10 orders of magnitude greater than SNPs [10]. Approximately 40–60 SNPs are required to discriminate as effectively as 13–15 short repeat polymorphisms [2]. For these reasons, we have chosen to use microsatellites over SNPs as the genetic marker to explore these various evolutionary models.

Chapter 3

Simulating Evolution

In this chapter we cover how we represent microsatellites, the different variations of simulation, and which simulation approach we take.

3.1 Microsatellite Representation

Let ℓ represent an integer associated with number of repeat units of a single microsatellite variation. For example if we define our nucleotide sequence as `GATA`, the microsatellite variation below would represented as $\ell = 4$.

$$\ldots \texttt{GATAGATAGATAGATA} \ldots$$

In addition to ℓ being an integer, we also constrain ℓ to exist between some lower bound on repeat units κ and some upper bound Ω. Any ℓ_M that meets the criteria in Equation 3.1 is defined to be the repeat unit associated with some variant for the microsatellite M:

$$(\ell_M \in [\kappa, \Omega]) \wedge \left(\ell_M \in \mathbb{Z}^+\right) \tag{3.1}$$

We denote this space to be $\mathbb{M}$:

$$\ell_M \in \mathbb{M} \tag{3.2}$$

Let $\pi^{(t)}$ represent an N-sized tuple, or microsatellite *population* of sequences $(\ell_1, \ell_2, \ldots \ell_N)$ in a human population of size $\frac{N}{2}$ at generation t. There exist two microsatellites variations per individual in a human population, but we relax the constraint that each human is associated with two sequences in $\pi^{(t)}$ for brevity. The unit of interest here is the microsatellite variation itself, which moves through generations while collecting or losing repeats until we reach the same generation our observations are recorded in.

3.2 Forward Simulation

There exists two main approaches toward which direction we should simulate toward: from the past to the present (forward) or from the present to the past (backward). In both cases, the goal is to create some evolutionary tree of n (distinct from N) microsatellites that trace

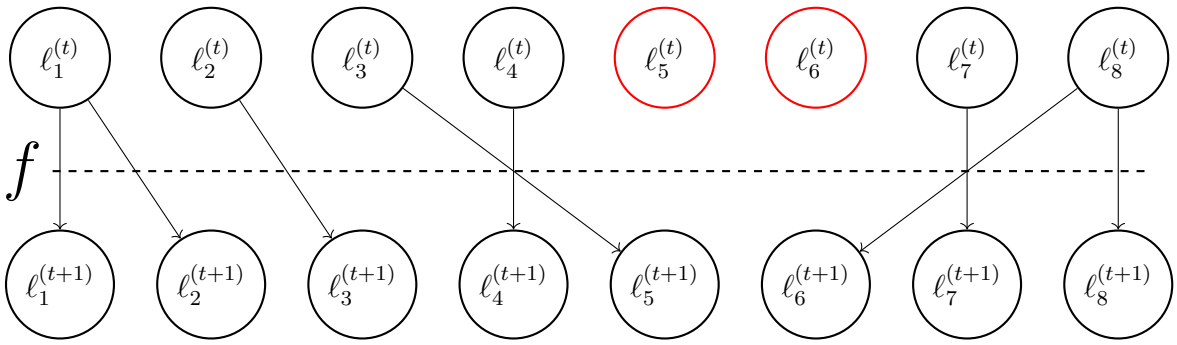

Figure 3.1: An example of population $\pi^{(t)}$ evolving to $\pi^{(t+1)}$ under the Wright-Fisher model. Each individual $\ell_i^{(t+1)} \in \pi^{(t+1)}$, represents the result of applying some mutation function f to some individual in $\ell_i^{(t)}$. Note that there are two individuals that do not advance to $\pi^{(t+1)}$: $\ell_5^{(t)}$ and $\ell_6^{(t)}$.

back to some common ancestor *efficiently*. Using these n generated microsatellite variants, we would then compare this to our n observed microsatellites. In the section, we focus on forward simulation– from past to present.

In order to simplify this problem, we assume the Wright-Fisher population model:

1. Our population of sequences is of constant size N for each generation.
2. All individuals from some generation t originate from the previous generation $t-1$.
3. There exists no selection bias, each sequence from some generation t is equally likely to be chosen to exist in the next generation $t+1$.

3.2.1 $\pi^{(t)}$ To $\pi^{(t+1)}$ Evolution

Let $\pi^{(t)}$ represent an effective microsatellite population at generation t, of size $\left|\pi^{(t)}\right| = N$. A forward simulation starts with population $\pi^{(1)}$, and evolves this until all members of the current population $\pi^{(\bar{t})}$ share a sole common ancestor. Starting from $\pi^{(1)}$, we chose N individuals from $\pi^{(1)}$ with replacement. For each individual chosen, we define the next generation $\pi^{(2)}$ as the output of this individual run through some mutation process f. f is defined as $f : \mathbb{M}, \mathbb{V} \to \mathbb{M}$, which takes some repeat unit and random variables $\mathcal{V}$ in space $\mathbb{V}$, and outputs another repeat unit. To generate $\pi^{(3)}$, we repeat the process to construct $\pi^{(2)}$ using $\pi^{(2)}$ instead of $\pi^{(1)}$ as our ancestor populace. This is repeated until we reach $\pi^{(\bar{t})}$.

An example of the Wright-Fisher evolution process is given in Figure 3.1. There exist two populations here: a parent population $\pi^{(t+1)}$ and a child population $\pi^{(t)}$, both of size $N = 8$. Note that two individuals from the parent population $\ell_5^{(t)}, \ell_6^{(t)}$ do not move to $\pi^{(t+1)}$, a consequence of choosing descendants *with replacement* from our ancestors.

3.2.2 Average Time Until n Individuals Coalesce: $\bar{t}$

The next question that follows is, "What is $\bar{t}$?". The naive approach (but most correct) is to keep track of the evolutionary history for all populations. This however requires (a)

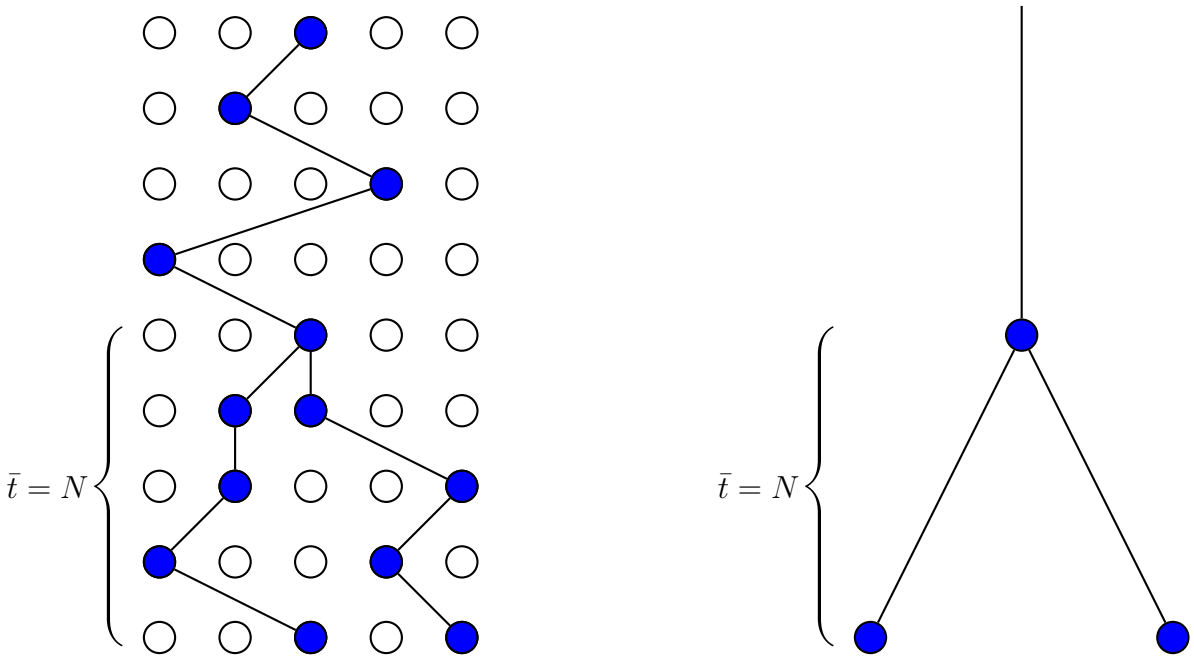

Figure 3.2: A coalescent buried in some forward simulation.

storage for the evolutionary tree formed and (b) an expensive search as we move forward for each generation. A compromise that avoids this search and storage increase is to fix $\bar{t}$ to some value, a time where most populations have a common ancestor shared across the entire population.

Let us start with $n = 2$ individuals and two generations: $t, t-1$. In a Wright-Fisher population, the probability $p(2, t-1)$ that any two individuals in $\pi^{(t)}$ share the same parent from $\pi^{(t-1)}$ is:

$$p(2, t-1) = \frac{1}{N} \tag{3.3}$$

Consequently, the probability $q(2, t-1)$ that two individuals do not share a parent from $\pi^{(t-1)}$ is:

$$q(2, t-1) = 1 - \frac{1}{N} \tag{3.4}$$

The probability that two individuals do not share a parent in $\pi^{(t-1)}$, but do have a common ancestor in $\pi^{(t-2)}$ is:

$$p(2, t-2) = q(2, t-1) \cdot \frac{1}{N} = \frac{1}{N} \cdot \left(1 - \frac{1}{N}\right) \tag{3.5}$$

If we ask the same question but for $\pi^{(t-3)}$, we get $p(2, t-3) = \frac{1}{N} \cdot \left(1 - \frac{1}{N}\right)^2$. Noting that this probability p is geometrically distributed, we can generalize p for having two individuals share a common ancestor y generations ago:

$$p(2, t-y) = \frac{1}{N} \cdot \left(1 - \frac{1}{N}\right)^{y-1} \tag{3.6}$$

with $p(2, t-y)$ having mean $E_2 = N$. The mean E_2 is interpreted as the expected number of generations until $n = 2$ individual *coalesce*.

Now we look at the probability that two individuals out of $n = 3$ individuals in $\pi^{(t)}$ share a parent from $\pi^{(t-1)}$. Here, we asking how many pairs can be formed with $n = 3$ out of the entire population:

$$p(3, t-1) = \frac{3}{N} \tag{3.7}$$

Given $n = 4$ individuals, the probability that any of the four share a parent in $\pi^{(t-1)}$ is $p(4, t-1) = \frac{6}{N}$. We generalize p for a coalescent event occurring with n individuals in the previous generation as such:

$$p(n, t-1) = \frac{\binom{n}{2}}{N} \tag{3.8}$$

For y generations, the general form of p is given [13]:

$$p(n, t-y) = \frac{\binom{n}{2}}{N} \cdot \left(1 - \frac{\binom{n}{2}}{N}\right)^{y-1} \tag{3.9}$$

with $p(n, t-y)$ having mean:

$$E_n = \frac{N}{\binom{n}{2}} \tag{3.10}$$

The mean E_n is interpreted as the expected number of generations until two out of n individuals coalesce.

Going back to our original question, what is the *average* time until n individuals share a common ancestor? Let us start with the average time until two individuals coalesce: $E_2 = N$ generations. The average time until three individuals coalesce is the average time until any two individuals coalesce *plus* the average time until coalescence occurs in any three individuals. For all n individuals, this average time constraint is defined in Equation 3.11. We approximate this further in Equation 3.12.

$$\bar{t} > \sum_{i=2}^{n} E_i \tag{3.11}$$

$$\bar{t} \gtrapprox 2N \tag{3.12}$$

3.2.3 Effective Population $\hat{\pi}$ Sampling

To circumvent the inefficient space strategy of tracking all lineages formed, we instead chose to simulate to a fixed number of generations $\bar{t} = 2N$. Let $\hat{\pi}$ represent a population of individuals who coalesce. We have to choose a select few individuals, or *sample* from $\hat{\pi}$ to obtain $\tilde{\pi}$. The data that we are comparing to our generated repeat units has been collected in samples, not entire populations. The problem here, is that our sample of n individuals may not share a common ancestor.

The simplest approach to mitigate the impact of this problem is to run our simulation process longer than the expected time to coalescence. This allows us to strengthen our belief

Algorithm 3.2.1: Generate a sample of individuals who *likely* share some common ancestors.

Function: *ForwardSimulator* $(\ell^{(1)}, N, n, t_\epsilon, f(\ell))$
  Input: a common ancestor $\ell^{(1)}$, population size N, sample size n, extra running time t_ϵ, a single-generation mutation function f, where $\mathcal{V}$ is implicit
  Output: a sample from the resultant population $\tilde{\pi}$
  $\pi^{(t-1)} \leftarrow \{\ell_1\}$, $\pi^{(t)} \leftarrow \emptyset$
  for $t \leftarrow 2$ **to** $t_\epsilon + 2N - 1$ **do**
    for $j \leftarrow 1$ **to** N **do**
      $\pi^{(t)} \leftarrow \pi^{(t)} \cup \{f(\text{a randomly selected } \ell \text{ from } \pi^{(t-1)})\}$
    end
    $\pi^{(t-1)} \leftarrow \pi^{(t)}$
    $\pi^{(t)} \leftarrow \emptyset$
  end
  $\tilde{\pi} \leftarrow n$ random ℓ from $\pi^{(t-1)}$
  return $\tilde{\pi}$

that all individuals in $\pi^{(2N)}$ share a common ancestor. We denote this additional running time as t_ϵ, a tunable parameter of our procedure. This makes our minimum running time:

$$\bar{t} = 2N + t_\epsilon \tag{3.13}$$

Once we have run the evolution process for $\bar{t}$ generations, we sample randomly n individuals.

3.2.4 Forward Evolution Algorithm

The function for forward evolution is presented in algorithm 3.2.1, and described below.

1. For simplicity, we treat our starting population $\pi^{(1)}$ as N instances of some common ancestor $\ell^{(1)}$. All individuals in the output π are expected to share one member from $\pi^{(1)}$, meaning that all other individuals in $\pi^{(1)}$ get thrown away. With this approach, we only need to worry about one parameter.

2. Mutation is discussed in detail in chapter 4. For now, we define $f(\ell)$ to be some function $f : \mathbb{M}, \mathbb{V} \to \mathbb{M}$ which takes some repeat unit and random variables, and returns another repeat unit.

3. We start by defining our storage sets, $\pi^{(t-1)}$ and $\pi^{(t)}$. These hold the ancestor and descendant generations respectively.

4. The simulation is run for $t_\epsilon + 2N$ generations. At each generation $t - 1$ we randomly sample N individuals, apply our mutation function f, and store the resultant in descendant generation $\pi^{(t)}$. The current descendant generation $\pi^{(t)}$ then becomes the next ancestor generation $\pi^{(t+1)}$ as we increment t.

5. Once we have reached $t_\epsilon + 2N$ iterations, we return n randomly sampled individuals from the last generation.

3.3 Backward Simulation

Forward simulation has the advantage of being incredibly straightforward to implement and apply complex generation to generation models to. The biggest disadvantages with this type of simulation though, are that (a) $2N^2$ individuals must be generated per simulation and (b) several returned populations from the forward simulator will not coalesce. In this section, we will go over backward simulation – an approach that addresses both of these problems.

3.3.1 Kingman's Coalescent Construction

As previously mentioned, the first drawback to forward simulation is the retention of individuals that do not exist in the evolutionary tree of the latest generation. Let $\mathcal{N}$ represent the total number of individuals we must simulate in order to produce a population sample of size n. For a population π of size $|\pi| = N$, we can compute the minimum number of total individuals that we simulate $\mathcal{N}^\star$:

$$\mathcal{N}^\star = \sum_{i=1}^{n} i = \binom{n+1}{2} \tag{3.14}$$

When using forward simulation with $\mathcal{N}_F = 2N^2$, there exists a waste of $\mathcal{N}_{waste}$ individuals created:

$$\begin{aligned} \mathcal{N}_{waste} &= \mathcal{N}_F - \mathcal{N}^\star \\ &= 2N^2 - \binom{n+1}{2} \end{aligned} \tag{3.15}$$

Given $N = 1,000, n = 100$, there exists $\mathcal{N}_{waste} = 1,994,950$. Nearly two-million individuals are needlessly generated to produce only 100 end individuals.

A backwards simulation reduces $\mathcal{N}_{waste}$ to 0 for all values of n by only working with individuals that contribute to the end population. Instead of working from generation to generation, this type of simulation works from one *coalescent event* to another. This allows us to abstract away from generational time steps (for now), and build a *coalescent tree*. The type of coalescent tree we are working with is known as a Kingman's coalescent. These follow a structure similar to Figure 3.3, varying only in which parent node connects two child nodes.

To start, we define $\pi_i^{(t)}$ as the repeat unit associated with some individual ℓ in population π at generation t. We define v_j as a node associated with this repeat unit, enumerated by j for a set of populations. Let V represent the set that holds all nodes used in an instance of a Kingman's coalescent. V can be thought of as all the nodes associated with the evolutionary tree, or graph, that we are building.

To state that some node v_a is the direct ancestor of v_b is to use the following *directed* edge:

$$(v_a, v_b) \Leftrightarrow v_a \text{ descends directly to } v_b \tag{3.16}$$

Building the set of directed weighted edges E that connect our graph requires the conditions given below.

1. For some edge (v_a, v_b), v_a must exist in a generation before v_b. This implies that v_1 is the source of our graph and all nodes in the generation-to-be-sampled $\hat{\pi}$ are sinks.

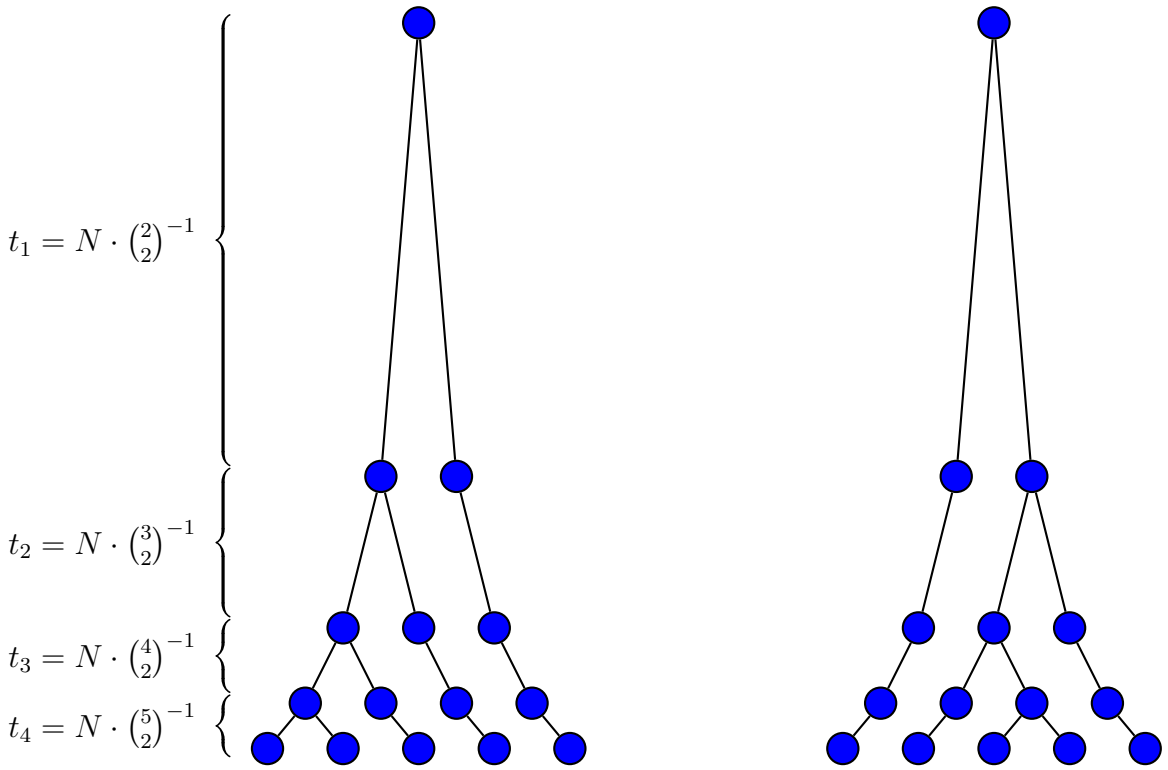

Figure 3.3: Two variations of a Kingman's coalescent tree with $n = 5$. Note the decrease in edge length (i.e. time) as the depth of nodes increases.

2. For some edge (v_a, v_b), both v_a and v_b must exist in the set of nodes associated with some instance of a Kingman's coalescent V.

3. Recall that all individuals of a Kingman's coalescent at generation t are copied into the following generation, with only one individual being copied twice. Let $V^{(t)}$ represent all nodes used in the Kingman's coalescent that exist in population $\pi^{(t)}$ at generation t. For all edges formed using $V^{(t+1)}$ and $V^{(t)}$, there exists only two edges $(v_a, v_b), (v_a, v_c)$ where the ancestor node v_a is seen twice. For all other edges the descendant node must be unique & from $V^{(t)}$, and the ancestor node must also be unique & from the set $V^{(t+1)}$. This constraint is illustrated with the coalescent tree examples in Figure 3.3.

4. Each edge (v_a, v_b) has an associated weight, representing the difference in generations (i.e. time) between nodes v_a and v_b. For an edge $e \in E$, this weight is found using the function $\tau : E, \mathbb{V} \rightarrow \mathbb{Z}^+$. Again, our random variables $\mathbb{V}$ indicate that the generation selected for each node is random.

3.3.2 Coalescent Repeat Unit Resolution

Constructing the tree is the first step associated with obtaining a sample of n individuals. The next step is to determine which repeat units are associated with some node $v_i \in V$. We

define the function $\lambda : E, \mathbb{V} \to \mathbb{M}$ which resolves the repeat unit of some individual of our coalescent tree. For all edges $(v_i, v_j) \in E$, the repeat unit ℓ_i associated with node v_i is given by running ℓ_i through f for a set number of generations to v_j. The next question of interest is, "How many generations do we apply f for?".

In section 3.2, we determined that the expected time between coalescent events for n individuals in a population of size N (Equation 3.10). This number naturally emerged as a result of the Wright-Fisher evolution process, but we must manually define this time if we are to use the coalescent abstraction. Let $\tau_N(e)$ represent a function $\tau_N : E, \mathbb{V} \to \mathbb{Z}^+$ that takes some edge e as input and returns the number of generations between the associated nodes. The most common approach is to draw these generations from an exponential distribution with mean $\frac{N}{\binom{\rho(e)}{2}}$ [13]:

$$e = \text{ an edge } (v_i, v_j) \text{ connecting nodes } v_i \text{ and } v_j \tag{3.17}$$

$$\tau_N(e) \sim \exp\left(\frac{N}{\binom{\rho(e)}{2}}\right) \tag{3.18}$$

where $\rho(e)$ represents an additional function $\rho : E \to \mathbb{Z}^+$ that determines the working coalescent event attached to some edge e. $\rho(e)$ can also be thought of as a depth determination function for the incoming edge attached to e.

Starting from the ancestor node v_1, we perform a breadth-first search (BFS) to visit nodes of the same depth first before moving to the next node. Upon arrival, we apply the mutation process to each visited node and descend further down our tree. This means that we perform our simulation in order of coalescent event, rather than lineage (which would require a depth first search, or DFS). Once we have visited each node, we return the repeat units associated with our leaves: $\{\lambda(v) \mid v \text{ is a leaf }\}$.

3.3.3 Backward Evolution Algorithm

The function for backward evolution is presented in algorithm 3.3.1, and described below:

1. There exist two stages here: the tree construction phase and the repeat unit determination phase. For the scope of this project it makes more sense to combine these phases and reduce our running time, however separating the two allows us to build more complex demographic models in the future (see chapter 8).

2. We use vector and index notation to represent our node and edge sets V, E. This is a more natural notation, and avoids having to explicitly specify λ. To specify consecutive vector elements (or *slices*), the $[a : b]$ notation is used. Here, a specifies the starting index of our vector (inclusive) and b specifies the end index of our vector (inclusive).

3. Contrary to the name "backward simulation", we build our tree top-down, or from ancestor to descendant. The working section of vectors V, E for a given iteration i is specified by the $i-1$'th and i'th triangle number. To build the edge vector, we randomly from $\left[\binom{i}{2}+1, \binom{i}{2}+2, \ldots, \binom{i+1}{2}\right]$ *without* replacement. This ensures that all ancestors advance to the next coalescent event. To replicate the coalescent event itself,

Algorithm 3.3.1: Generate a sample of individuals who share a common ancestor from some effective population.

1 **Function:** *TwoStageBackwardSimulator* $(\ell^{(1)}, n, f, \tau_N)$
  Input: a common ancestor $\ell^{(1)}$, sample size n, a single-generation mutation function f where $\mathcal{V}$ is implicit, a generation determination function τ_N where $\mathcal{V}$ is implicit
  Output: a sample from the resultant population $\tilde{\pi}$
2 $V \leftarrow$ an empty $\binom{2N}{2}$ sized vector, $E \leftarrow$ an empty $(\binom{2N}{2} - 1)$ sized vector
3 **for** $i \leftarrow 2$ **to** $2n$ **do** `/* Stage 1: Construct the tree. */`
4 $E' \leftarrow$ an empty i sized vector, $A \leftarrow \{a \mid a \in [\binom{i-1}{2} + 1, \binom{i}{2}] \wedge i \in \mathbb{Z}\}$
5 **for** $j \leftarrow \binom{i-1}{2} + 1$ **to** $\binom{i}{2}$ **do**
6 $E'[j] \leftarrow (V[i], V[a'])$, where a' is randomly selected from A
7 $A \leftarrow A - \{a'\}$
8 **end**
9 $E'[i+1] \leftarrow (V[i+1], V[a'])$, where a' is randomly selected from $\{a \mid a \in [1, \binom{i}{2}] \wedge i \in \mathbb{Z}\}$
10 $E[\binom{i}{2} : \binom{i+1}{2}] \leftarrow \hat{E}$
11 **end**
12 $V[1] \leftarrow \ell_1$
13 **for** $i \leftarrow 2$ **to** $|V|$ **do** `/* Stage 2: Determine the repeat units. */`
14 $v' \leftarrow$ the outgoing node attached to $E[i]$
15 **for** $j \leftarrow 1$ **to** $\tau_N(e)$ **do**
16 $v' \leftarrow f(v')$
17 **end**
18 $V[i+1] \leftarrow v'$
19 **end**
20 **return** $V[\binom{n-1}{2} + 1 : \binom{n}{2}]$

we randomly select one ancestor to yield two descendants by generating an additional edge (see line 9).

4. To determine the repeat unit (i.e. populate V), we move back to top and evolve down. Starting from the given common ancestor, we perform BFS by visiting the incoming nodes attached to edges in the order they were created in. Maintaining this order allows us to avoid having to specify the depth determination function $\rho(e)$. For each outgoing node v' attached to edge e, we apply our mutation process $\tau_N(e)$ times. The result is saved to the incoming node attached to e. This is repeated for all edges.

5. We return the leaves associated with our tree: the last n elements of the V vector.

Chapter 4

Modeling Mutation

In this chapter, we discuss mutation and describe our proposed 2-parameter $\theta = \{d, c\}$ mutation model.

4.1 Repeat Unit Change Per Generation

Mutation refers to a change in DNA sequence. For this paper, we refer to mutation as a change in the repeat unit of some locus for a microsatellite variant. We are also only discussing mutations that are passed on to the next generation, under the neutral (Wright-Fisher) assumptions.

The simplest model of microsatellite mutation and evolution is the stepwise mutation model of Ohta and Kimura [17]. Let $f : \mathbb{M}, \mathbb{V} \to \mathbb{M}$ represent some function that accepts some repeat unit and random variables $\mathcal{V} \in \mathbb{V}$, and outputs some repeat unit. Given a population $\pi^{(t)}$ at generation t and some repeat unit $\ell^{(t)} \in \pi^{(t)}$, there are three possibilites for the mutated repeat unit in the next generation $f(\ell^{(t)}) = \ell^{(t+1)}$:

$$f(\ell^{(t)}) \in \{\ell^{(t)} + 1, \ell^{(t)}, \ell^{(t)} - 1\} \tag{4.1}$$

Per generation, each microsatellite variant mutates up one repeat unit (expansion), down one repeat unit (contraction), or stays the same – all with equal probability.

Let μ_u represent the probability of upward mutation and μ_d represent the probability of downward mutation. We can construct our function f as such:

1. We are given some repeat unit $\ell^{(t)}$, an upward mutation rate μ_u, and a downward mutation rate μ_d.
2. Sample two numbers from a uniform random distribution: $x_u, x_d \sim U(0, 1)$.
3. If $\mu_u > x_u$, then $f(\ell^{(t)}) = \ell^{(t)} + 1$. This is the inverse transform sampling method.
4. If $\mu_d > x_d$ but $\mu_u \leq x_u$, then $f(\ell^{(t)}) = \ell^{(t)} - 1$.
5. If $(\mu_d > x_d \ \wedge \ \mu_u > x_u) \ \vee \ (\mu_d < x_d \ \wedge \ \mu_u < x_u)$, then $f(\ell^{(t)}) = \ell^{(t)}$.

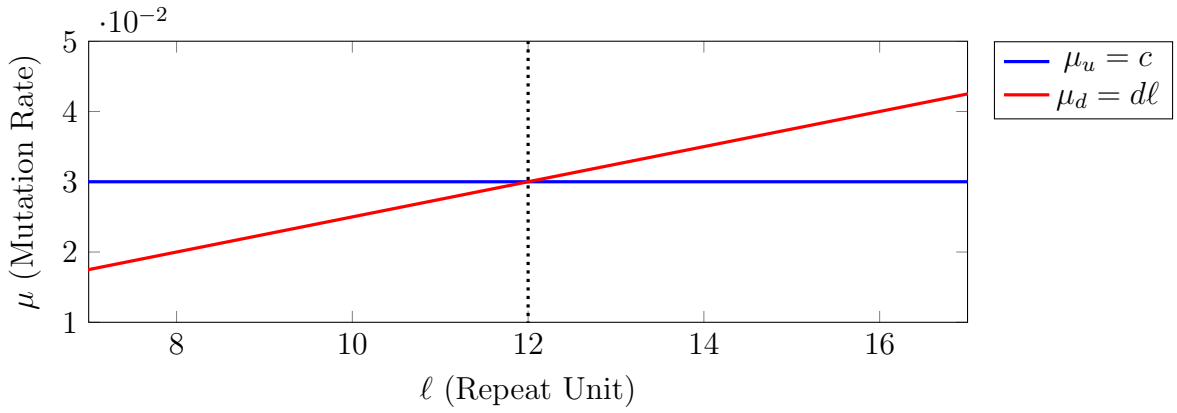

Figure 4.1: Our mutation model for $d = 0.0025, c = 0.03$. Our focal length with these parameters is $\hat{\ell} = 12$.

There does exist evidence that mutations occur at steps greater than one repeat unit. A model that takes this into account is the two-phase model, which follows the stepwise mutation model with probability p and mutates up and down with a unit sampled from a geometric distribution, with probability $1 - p$ [6]. This is modeled by the procedure below:

1. We are given some repeat unit $\ell^{(t)}$, an upward mutation rate μ_u, a downward mutation rate μ_d, the probability of the stepwise mutation model p, and the success probability for the geometric distribution m.
2. Sample three numbers from a uniform random distribution: $x_u, x_d, x_p \sim U(0, 1)$.
3. If $p > x_p$, then we set the changes in repeat unit $\Delta_u = 1, \Delta_d = 1$. Otherwise, we sample our changes from a geometric distribution: $\Delta_u, \Delta_d \sim Geom(m)$.
4. If $\mu_u > x_u$, then $f(\ell^{(t)}) = \ell^{(t)} + \Delta_u$.
5. If $\mu_d > x_d \wedge \mu_u \leq x_u$, then $f(\ell^{(t)}) = \ell^{(t)} - \Delta_d$.
6. If $(\mu_d > x_d \ \wedge \ \mu_u > x_u) \ \vee \ (\mu_d < x_d \ \wedge \ \mu_u < x_u)$, then $f(\ell^{(t)}) = \ell^{(t)} + \Delta_u - \Delta_d$.

The two-phase model is more flexible here, but this introduces two new parameters to estimate: p and m. According to Sainudiin, the likelihood of all "one phase" models are not significantly different than their two phase counterparts [20]. For this reason, our model restricts repeat units to mutate by at most one per generation.

4.2 Mutation Rate Dependence on Repeat Unit

An assumption made with stepwise mutation model is that the probability of mutation is equal for all repeat units. Ellegren showed that allele unit plays a role in how often mutations occur [7]. More specifically, longer repeat units mutate more often than shorter repeat units. To introduce this repeat unit dependence, we use the following equation:

$$\mu = s \cdot \ell \tag{4.2}$$

Algorithm 4.2.1: The algorithmic description for $f : \mathbb{M}, \mathbb{V} \to \mathbb{M}$. To stay consistent with the definition of f we denote c and d as parameters of the model itself.

Function: *MutateFunction* $(c, d, \ell^{(t)})$
  Input: an upward constant mutation bias c, downward linear mutation bias d, current repeat unit $\ell^{(t)}$
  Output: a repeat unit from the next generation $\ell^{(t+1)}$
  $\ell^{(t+1)} \leftarrow \ell^{(t)}$
  if $\ell^{(t+1)} = \kappa$ **then**
    return $\ell^{(t+1)}$
  end
  if $c < \sim U(0, 1)$ **then**
    $\ell^{(t+1)} \leftarrow \ell^{(t+1)} + 1$
  end
  if $d\ell < \sim U(0, 1)$ **then**
    $\ell^{(t+1)} \leftarrow \ell^{(t+1)} - 1$
  end
  return $\ell^{(t+1)}$

where s represents the strength of this dependence. There exists evidence for more complex models such as the exponentially increasing mutation rate of Xu et. al. [26]. In this case, our repeat unit dependence would be represented by:

$$\mu = s \cdot \exp(\ell) \tag{4.3}$$

Following Sainudiin's general model which included only linear dependence, we made the decision to use the first equation (Equation 4.2).

4.3 Presence of Mutational Bias

There now exists a mutation rate dependence on repeat unit, but the probability of mutating up or down is still equally likely. Garza's observed that microsatellites tend to mutate toward a *focal unit* [9]. If the repeat unit of interest ℓ is below the focal length at some generation t, ℓ will have a higher probability of mutating upward than downward in the following generations. Conversely if ℓ is above the focal length at some generation t, ℓ will have a higher probability of mutating downward than upward in the following generations.

Sainudiin's general model uses the functions $\beta : \ell, s \mapsto [0, 1]$ and $\alpha : u, v, \ell \mapsto [0, 1]$ to establish these biases:

$$\beta(\ell, s) = \mu(1 + (\ell - \kappa)s) \tag{4.4}$$

$$\alpha(u, v, \ell) = \max(0, \min(1, u - v(\ell - \kappa))) \tag{4.5}$$

where β is the rate of mutation and α is probability that our repeat unit is incremented (expanded). Here u represents some constant bias applied toward expansion and v represents the linear bias toward expansion. A focal unit $\hat{\ell} = ((u - 0.5)/v)$ is specified if we have a

greater tendency toward expansion at lower repeat units ($0.5 < u < 1$) and if the following holds: $(u - 0.5)/(\Omega - \kappa) < v < \infty$ [20]. In other words, a focal unit is defined where the probability of expansion equals the probability of contraction.

4.4 Proposed Mutation Model

This now brings us to our complete mutation model. To introduce a bias toward some focal unit, we define two mutation rates: the upward mutation rate μ_u and the downward mutation rate μ_d. The total mutation rate is the sum of the two: $\mu = \mu_u + \mu_d$. We define μ_d to be dependent on repeat unit, and μ_u to be constant:

$$\mu_u = c \tag{4.6}$$

$$\mu_d = d\ell \tag{4.7}$$

where c represents a constant bias for the upward mutation (or, the upward mutation rate itself) and d represents a linear bias for the downward mutation rate. We constrain $c > 0, d \geq 0$ to ensure that mutation is always occurring. The focal unit $\hat{\ell}$ is defined as the intersection between the upward and downward mutation rate, where the probability of expansion equals to the probability of contraction.

$$\hat{\ell} = \frac{c}{d} \tag{4.8}$$

This differs from Sainudiin's model usage of $\alpha(u, v, \ell)$, but for our purposes is functionally equivalent and requires two less parameters. In addition to the three requirements listed in the previous sections, we constrain that any individuals having repeat unit $\ell = \kappa$ are unable to mutate back up. If f is given some repeat unit ℓ equal to the lower bound of $\mathbb{M}$, ℓ is returned. This is meant to simulate how microsatellites are lost in populations.

Graphically, this is represented in Figure 4.1. We describe the entire mutation function f in algorithm 4.2.1. We achieve the first state $\ell^{(t)} + 1$ for some $\ell^{(t)}$ by applying our upward mutation first and saving this to $\ell^{(t+1)}$. We achieve states $\ell^{(t)} - 1$ *and* $\ell^{(t)}$ by applying our downward mutation second, after performing our upward mutation. Only after performing these two in sequence do we return our mutated repeat unit $\ell^{(t+1)}$.

Chapter 5

Data for Parameter Estimation

In this chapter, we discuss the observed data we are working with. In the past several sections, we have worked on artificially generating a population of repeat units given several parameters. The data that we collect through observation must be in a similar format to this generated population. This allows us to determine the likelihood of some parameter set (see chapter 6).

The data we are working with comes from ALFRED, the ALlele FREquency Database of humans from Yale. The steps to extract the data from the ALFRED website are as follows:

1. Navigate to https://alfred.med.yale.edu/alfred/index.asp.
2. Type "GATA" into the search box to search for `GATA` repeats. We are using tetranucleotide repeats instead of other microsatellites because tetranucleotide repeats mutate more often than dinucleotide repeats [23].
3. For each result, we saved the data if it was packaged in a frequency vs. repeat unit format. Given a sample size n, this data can easily be transformed into a population of repeat units. The inverse is also true (and is closer to what was performed): given a population of repeat units, a collection of frequency against repeat units is easy to obtain as well.

This gave us over samples from over 147 distinct populations. For this research, we are only the population with the highest number of sample sets: the Columbians. The Columbian populace was used as training data to determine the c, d parameters. There exists plans to utilize more of the data we have collected from ALFRED for additional training, verification, and testing.

Chapter 6

Parameter Estimation

In this chapter, we discuss our approach toward parameter estimation. Specifically, we aim to address the problem of maximum likelihood.

6.1 Intractable Likelihood Functions

We denote θ to be a set of parameters values, Θ to be the space all of our parameters values reside in, $\mathcal{D}$ to be a collection of observed data, and $\mathcal{D}'$ to be a collection of generated data. The *likelihood* of some parameter values $\theta \in \Theta$ is the joint probability that θ produces some observed data $\mathcal{D}$. Let $\mathcal{M}, \mathcal{V} : \theta \mapsto \mathcal{D}'$ define a function that maps parameter values θ and random variables $\mathcal{V}$ to generated data $\mathcal{D}'$. Given that $\mathcal{V}$ is random, different $\mathcal{D}'$ may be returned for the same parameter values θ. Given *discrete and countable* observations $\mathcal{D}$, the likelihood of parameter values $\theta \in \Theta$ is the probability that our generated data equals our observed data [15].

$$\mathcal{L}(\theta) = \Pr(\mathcal{D} = \mathcal{M}(\theta)) \tag{6.1}$$

One technique to calculate a maximum likelihood for our parameter values involves finding the critical points of $\mathcal{L}$ (where $\frac{d\mathcal{L}}{d\theta} = 0$) and choosing the point $\hat{\theta}$ with the largest value of $\mathcal{L}$. In Figure 6.1, the maximum of such a likelihood is displayed at the intersection of the blue and black dashed lines. There are however, several problems with this approach:

1. A large range $\mathcal{D}'$ for simulator $\mathcal{M}$ indicates that the frequency of exact observed – generated matches will be too low to make inferences.

2. This assumes that we can explicitly express $\mathcal{L}$. For simulator based $\mathcal{M}$ like ours, this is not trivial to do.

Consequently, we must look into other approaches to *infer* $\mathcal{L}$. The two problems this chapter aims to address are (a) how to efficiently compute $\mathcal{L}$ for a single θ and (b) how to infer $\mathcal{L}$ for all θ.

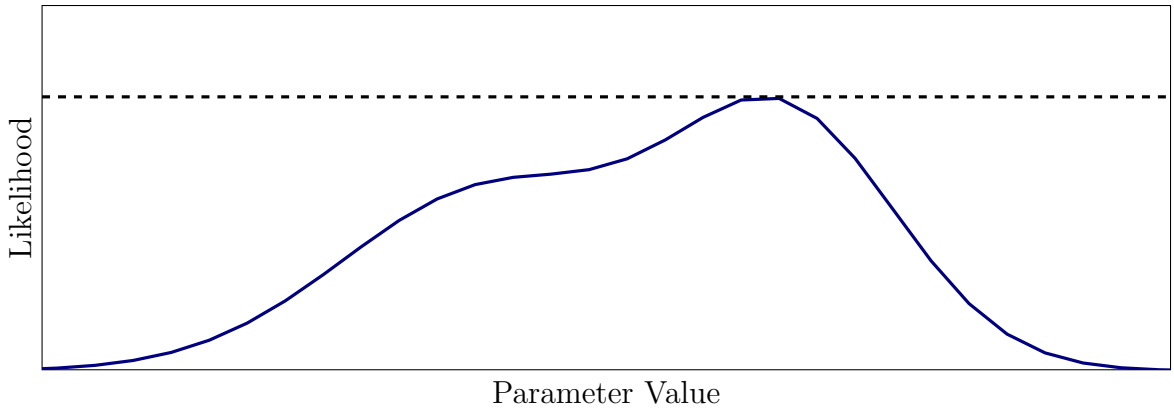

Figure 6.1: General figure depicting a smooth likelihood distribution (blue). The maximum likelihood point of this distribution is the intersection of the black dashed line and the blue line.

6.2 Approximate Bayesian Computation

In this section, we discuss an approximate method to compute the likelihood $\mathcal{L}$ for some parameter values θ.

As previously mentioned, to find $\mathcal{L}$ is to find the joint probability that our model and parameters produce the observed data. We expand Equation 6.1 to make this explicit for r sets of observed data in Equation 6.2. θ remains the same, however the randomness of $\mathcal{V}$ may result in different probabilities. For brevity, $\mathcal{V}$ is implicit in $\mathcal{M}$ from this point forward.

$$\mathcal{L}(\theta) = \prod_{i=1}^{r} \Pr\left(\mathcal{M}(\theta) = \mathcal{D}_i\right) \tag{6.2}$$

Let $\mathcal{D}$ represent the set of observed samples from the Columbian populace. How do we determine the probability of our model generating some $\mathcal{D}_i \in \mathcal{D}$ to determine $\mathcal{L}(\theta)$? Here, we take a frequentist approach and perform the following steps:

1. Run our simulator once to get $\mathcal{D}'_1$.
2. Check if $\mathcal{D}'_1$ matches $\mathcal{D}_1$.
3. Repeat steps 1 and 2 some number of times T_1, for different simulated samples but the same observed sample. We define $\Pr\left(\mathcal{M}(\theta) = \mathcal{D}_1\right)$ as the frequency of exact matches.
4. Repeat step 3 for all observed samples to get $\Pr\left(\mathcal{M}(\theta) = \mathcal{D}_2\right)$, $\ldots$, $\Pr\left(\mathcal{M}(\theta) = \mathcal{D}_r\right)$.
5. To find the $\mathcal{L}$ is to multiply all r probabilities together.

This approach seems simple enough, but there exists one caveat: the frequency of exact matches is too low to interpret anything meaningful. The microsatellite repeat length set $\mathbb{M}$ consists of roughly 30 elements, meaning that roughly 30 dimensions must exactly match. In addition to this, each frequency resides in some large set whose elements are between $[0, 1]$. The solution we propose here is a technique known as *Approximate Bayesian Computation*,

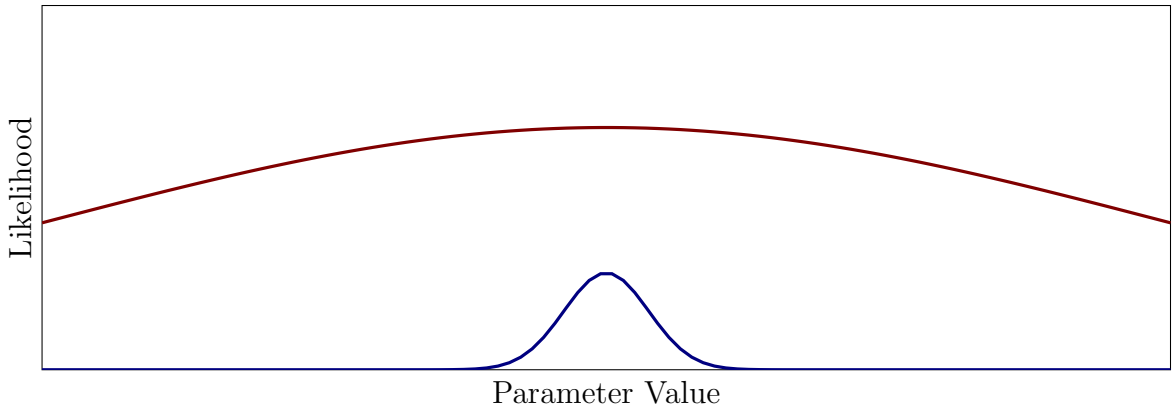

Figure 6.2: General figure depicting a true likelihood surface (blue) and an approximate surface (red). The true likelihood may not be found in a reasonable amount time, whereas the approximate likelihood is wider but taller (making it more tractable, but less defined).

or ABC for short. ABC has two parts: the use of *approximate* matches and the use of summary statistics [15]. In the following sections, I discuss how we define approximate, what summary statistics are, and why we are not using them for this problem.

6.2.1 Approximate Matches: ϵ

Let us dissect $\mathcal{D}$ and $\mathcal{D}'$ further. A given $\mathcal{D}_i \in \mathcal{D}$ and $\mathcal{D}'_i \in \mathcal{D}'$ define $|\mathbb{M}|$-sized tuples, whose values exist in $[0, 1]$. To compare a given $\mathcal{D}_i$ and $\mathcal{D}'_i$ is to iterate through each element (repeat unit) in both tuples and measure their proximity to each other. Let $\delta : \mathcal{D}_i, \mathcal{D}'_i \mapsto [0, 1]$ represent some function that accepts the observed and generated sample, and outputs some distance between 0 and 1. The comparison between our observed and generated samples is given by $\delta(\mathcal{D}_i, \mathcal{D}'_i)$.

We were only able to explore one δ function: the angular (or Cosine) distance. The angular distance treats $\mathcal{D}_i$ and $\mathcal{D}'_i$ as $|\mathbb{M}|$-dimensional vectors and aims to quantify some difference between the two. An output of 0 indicates that both samples are completely similar, while an output of 1 indicates that two vectors are maximally dissimilar (orthogonal). The angular distance δ_A is defined below [4]:

$$\delta_A(\mathcal{D}_i, \mathcal{D}'_i) = \frac{2}{\pi} \arccos\left(\frac{\sum_{\ell=\kappa}^{\Omega} \mathcal{D}_i[\ell] \cdot \mathcal{D}'_i[\ell]}{\sqrt{\sum_{\ell=\kappa}^{\Omega} (\mathcal{D}_i[\ell])^2} \cdot \sqrt{\sum_{\ell=\kappa}^{\Omega} (\mathcal{D}'_i[\ell])^2}} \right) \tag{6.3}$$

where $\mathcal{D}_i[\ell]$ and $\mathcal{D}'_i[\ell]$ represent the frequency of repeat length ℓ for the observed and generated samples respectively.

With some distance is quantified, the next step is defining what "approximate" means. According to ABC, two samples $\mathcal{D}_i$ and $\mathcal{D}'_i$ are approximate matches if their distance falls below some threshold $\epsilon \in [0, 1]$ [16]:

$$(\delta(\mathcal{D}_i, \mathcal{D}'_i) < \epsilon) \Leftrightarrow (\mathcal{D}_i \text{ and } \mathcal{D}'_i \text{ are approximate matches}) \tag{6.4}$$

By increasing ϵ, the frequency of exact observed – generated matches increases as well. This results in a flatter curve, as seen in Figure 6.2.

The next question that follows is "How do we know which ϵ to use?" If ϵ is too small, the problem becomes intractable. If ϵ is too large, we draw values that are not representative of the original distribution. There is no clear answer to this question, and Lintusaari et. al. states this choice is typically made by experimenting with different $(\mathcal{D}, \theta)$ pairs [15]. We define our threshold ϵ as a *hyperparameter*, a parameter we must specify and often experiment with) to find the parameters of interest θ.

6.2.2 Dimension Reduction: Summary Statistics

An alternative to using a distance function δ that deals with $|\mathbb{M}|$-dimensional tuples is to use a function that reduces, or *summarizes* the data into two points $\bar{\mathcal{D}}, \bar{\mathcal{D}}'$ of lower dimensionality and finds a distance between both $\bar{\mathcal{D}}, \bar{\mathcal{D}}'$. We specify a distance function δ_S that performs this transformation using functions $h : \mathcal{D}_i \mapsto \bar{\mathcal{D}}_i$ and $h' : \mathcal{D}'_i \mapsto \bar{\mathcal{D}}'_i$ as such:

$$\bar{\delta_S}(\mathcal{D}_i, \mathcal{D}'_i) = \bar{\delta}(h(\mathcal{D}_i), h'(\mathcal{D}'_i)) \tag{6.5}$$

A common choice for h, h' is the mean or median. For us, it may make sense to use the focal unit computation $\hat{\ell}$ as our summary statistic.

Using summary statistics avoids the curse of dimensionality (see [1]) for high dimension distance functions, but this adds yet another item we must specify: "Which summary statistic is the best?". If we summarize our data wrong, we again run into the problem of drawing values that do not represent our original distribution. We ran several trials without reducing our dimensionality and have not run into any problems thus far. To reduce noise, we are only using the angular distance δ_A for $|\mathbb{M}|$-dimensional vectors.

6.3 Markov Chain Monte Carlo

In this section, we discuss the Markov Chain Monte Carlo (MCMC) approach to approximating a likelihood function.

6.3.1 Monte Carlo

In section 6.2, we explored how to determine the likelihood of a single point θ. We are now interested in the most likely θ out of all possible parameter values Θ. We considered a naive approach to determining the most likely point $\hat{\theta}$ in section 6.1, which involved determining the derivative of some function we can explicitly express. Given that we cannot express our function as such, this is not an option. The solution Monte Carlo algorithms propose is choosing θ randomly and select the most likely θ out of all runs.

Let p define a probability distribution that determines how we draw θ. p is more commonly known as a *prior distribution*, and allows us to insert any prior beliefs we have about our likelihood before finding $\mathcal{L}(\theta)$ itself. In Bayesian inference, the characterization of the uncertainty of some θ given observations $\mathcal{D}$ is given by another distribution known as the

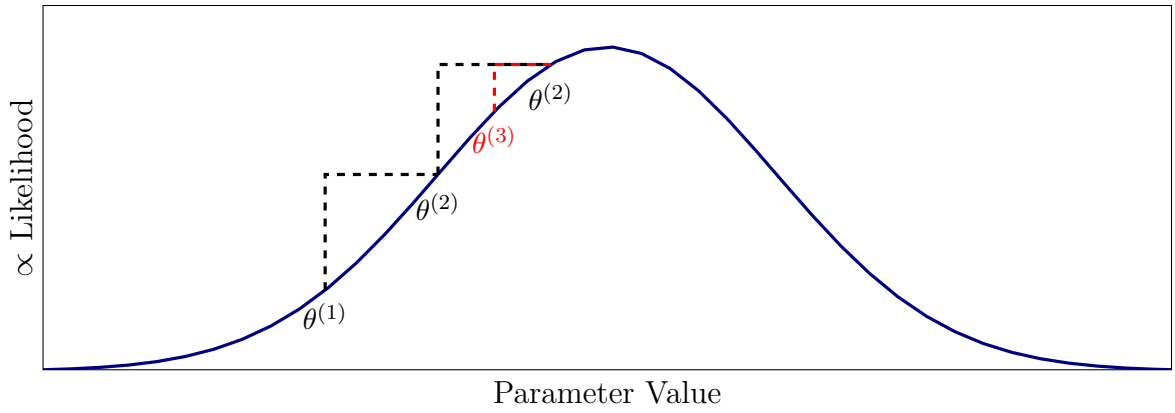

Figure 6.3: General figure depicting the random walk of Metropolis sampler around some surface poportional to our likelihood (the posterior). We start at initial state $\theta^{(1)}$. We accept $\theta^{(2)}$ and $\theta^{(3)}$ which leads to regions of higher likelihood but later accept $\theta^{(4)}$, a less likely point due to the randomness of $\mathcal{V}$.

posterior distribution [15]:

$$\Pr(\theta \mid \mathcal{D}) \propto \mathcal{L}(\theta)p(\theta) \tag{6.6}$$

According to Baye's law, the posterior is proportional to likelihood function we are trying to find. If we can explicitly express our posterior, then the $\hat{\theta}$ that maximizes $\Pr(\theta \mid \mathcal{D})$ also maximizes $\mathcal{L}(\theta)$.

6.3.2 Markov Chains

With the Monte Carlo strategy, our general procedure now becomes:

1. Draw θ_i from our prior $p(\theta)$.
2. Determine the likelihood of this point $\mathcal{L}(\theta_i)$.
3. Repeat until we have a representative set of samples.
4. Fit our samples to a curve, and return the $\hat{\theta}$ that maximizes $\mathcal{L}(\theta)$.

The main problem with this approach is our reliance on the prior. A misinformed prior will produce samples that are not representative of the posterior. As an example if $E(p(\theta)) = 500$ but our posterior is centered around $\theta = 1$, we will end up with a small or nonexistent posterior. *Markov Chain Monte Carlo* methods solve this by sampling more often from regions of higher likelihood.

Markov Chain Monte Carlo methods work by constructing a Markov chain such that the posterior distribution is its equilibrium distribution. We start by defining a Θ-sized vector Y, indexed by all distinct $\theta \in \Theta$ and whose values represent probabilities associated with each θ. Y is said to be a distribution here. Next we define a matrix G of size $\Theta \times \Theta$, whose entries describe the probability of transitioning from one θ (row) to another θ (column). G

is known as a *transition matrix*. We are able to move from distribution $Y^{(i)}$ to $Y^{(i+1)}$ using G:

$$Y^{(i+1)} = Y^{(i)}G \tag{6.7}$$

Y is said to be at *equilibrium* if the following holds true for some transition matrix G:

$$Y^{(i)} = Y^{(i)}G \tag{6.8}$$

The goal of MCMC is to draw samples X from Y such that Y represents our posterior $\Pr(\theta \mid \mathcal{D})$. Let X represent this *chain* of states, or parameter values θ, that satisfy the conditions below to draw from such a distribution [12]:

1. *X is a Markov chain.* The probability of obtaining $X^{(i)} \in X$ from $X^{(i-1)} \in X$ does not depend on any other configuration $X^{(i-2)}, X^{(i-3)}, \ldots, X^{(1)}$ other than $X^{(i-1)}$ itself.

2. *X is irreducible.* This states that we are able to travel to all of Θ from any given θ in a finite number of transitions.

3. *X is aperiodic for all configurations.* A state θ is aperiodic if there exists a i such that for all $j \geq i$:
$$\Pr\left(X^{(j)} = \theta \mid X^{(1)} = \theta\right) > 0 \tag{6.9}$$

4. *X is positive recurrent.* The states that the expected number of transitions to move to back to the same state is finite.

MCMC comprises a class of algorithms that are able to produce X that satisfy these conditions.

6.3.3 Metropolis Algorithm

As per our last section, we want to generate some chain of states X such that the given conditions are satisfied. In this section, we describe the Metropolis algorithm– a procedure that is able to generate a chain of states such that these conditions above are met. There are three main steps to the Metropolis algorithm:

1. *Proposal*: We define some function $g : \Theta, \mathbb{V} \to \Theta$ which generates a new and random θ_i given an old θ'. This relates to the transition matrix G from subsection 6.3.2, and constructs different G for different values of θ_i. The Metropolis algorithm is a special case of the *Metropolis-Hastings algorithm* in which this proposal is *symmetric*. A symmetric proposal indicates that the probability of drawing our new value given our current value is equal to the probability of drawing our current value given our new value.

2. *Calculate*: We determine the acceptance ratio α, which is a ratio of the proposed $\theta^{(i)}$ to the old θ'.
$$\alpha = \frac{\mathcal{L}(\theta_i)}{\mathcal{L}(\theta)} \tag{6.10}$$

Algorithm 6.3.1: The Metropolis algorithm, used to produce samples from a posterior distribution proportional to our likelihood.

Function: *MetropolisSampler* $(T_2, \mathcal{L}, \theta^{(1)}, g)$
  Input: number of algorithm iterations T_2, likelihood function $\mathcal{L}$, initial state θ_1, proposal function g
  Output: samples from our posterior distribution $\Pr(\theta \mid \mathcal{D})$
  $X \leftarrow \emptyset$, $\theta' \leftarrow \theta^{(1)}$
  for $i \leftarrow 1$ **to** T_2 **do**
    $\theta^{(i)} \leftarrow g(\theta')$
    if $\mathcal{L}(\theta^{(i)}) \cdot \mathcal{L}(\theta')^{-1} \leq \sim U(0,1)$ **then**
      $\theta' \leftarrow \theta_i$, $X \leftarrow X \cup \{\theta^{(i)}\}$
    end
  end
  return X

3. *Accept*: We save θ_i and $\mathcal{L}(\theta_i)$ to our collection of states if and only if θ_i is more likely than the old θ' *or* the ratio of proposed to old likelihoods is greater than some uniform random variable $U(0,1)$. If this is not true, then we go back to step (1) until we have run T_2 iterations.

Note that we have introduced three new hyperparameters here: our initial Markov chain position $\theta^{(1)}$, our proposal function g, and the number of Metropolis sampler iterations T_2. If $\theta^{(1)}$ is nowhere near the regions of high likelihood, if g is too tightly or loosely distributed, or if we do not run our sampler for enough iterations, we say that our Markov chain has not *converged*. Nonconvergence means that we cannot interpret anything meaningful from X, making the selection of these hyperparameters *and* some convergence verification process critical.

6.4 Maximum Likelihood Estimation

In this section, we explore a high level view of our approach to maximum likelihood estimation. Using the Metropolis sampler, we are able to get a collection of states X from a distribution proportional to our likelihood. We now want to tie this back into our original question: "Which parameter values $\theta \in \Theta$ are the most likely to produce our observations $\mathcal{D}$?". Given states X from our posterior, we determine this most likely point θ by (a) constructing histograms, (b) fitting the histograms to the equation of some distribution, and (c) determining the mean of this distribution.

We start by constructing our histogram. For this project, each θ represents a 2-tuple of c and d. We construct sets C, D which consist of each c and d element for all $\theta \in X$:

$$C = \{c \mid c \in \theta \wedge \theta \in X\} \tag{6.11}$$

$$D = \{d \mid d \in \theta \wedge \theta \in X\} \tag{6.12}$$

We now want to partition each set C and D sets into consecutive, non-overlapping intervals or *bins* $C^\star$ and $D^\star$ respectively. The partitioning itself is governed by the boundaries of our results $[\min(C), \max(C)]$, $[\min(D), \max(D)]$, and the *bin widths* b_c and b_d for the c and d parameters respectively. This gives us the number of bins:

$$\begin{aligned} |C^\star| &= \left\lceil \frac{\max(C) - \min(C)}{b_c} \right\rceil \\ |D^\star| &= \left\lceil \frac{\max(D) - \min(D)}{b_d} \right\rceil \end{aligned} \tag{6.13}$$

We build our histogram for c by constructing the function $w_c : [0, |C^\star|] \to [0, 1]$ that accepts an integer that enumerates our bins and outputs the frequency of associated with that bin. The end result must follow the property in Equation 6.14 to ensure our histogram represents some probability density function.

$$\int_{i=C^\star} w(i)di = 1 \tag{6.14}$$

The next step is to fit our histogram w_c to some known distribution. We explored two distributions here: the normal distribution and the gamma distribution. If we assume that our posterior for some C is normally distributed, we can use the function below:

$$w_{cN}(c) = \frac{\exp(\frac{((c\ -\ loc)/scale)^2}{2})}{scale \cdot \sqrt{2\pi}} \tag{6.15}$$

To get the point of maximum likelihood point $\hat{c}$ using $w_N(c)$ is to obtain its mean. For the distribution in Equation 6.15, this is the *loc* parameter with variance *scale*. If we instead assume our posterior for some C is gamma distributed, we can use the function below:

$$w_{c\Gamma}(c) = \frac{(x - loc)^{a-1} \exp(-\frac{c - loc}{scale})}{scale^a \cdot \Gamma(a)} \tag{6.16}$$

where *loc* represents a horizontal shifting parameter, *scale* represents the scaling parameter of our distribution, and a represents the gamma distribution *shape* or skew parameter. Again, the maximum likelihood $\hat{c}$ using this $w_\Gamma(c)$ is the mean. For the distribution given above, this is equal to $a \cdot scale + loc$ with variance $a \cdot scale^2$. For both cases, these steps are then repeated for D to obtain $\hat{d}$.

Chapter 7

Discussion

In this chapter, we discuss our implementation of our ABC-MCMC approach and our results.

7.1 Implementation

All algorithms mentioned in this paper were implemented in Python 3.7. To optimize the coalescent tree generation, repeat length determination, and likelihood determination processes, the Numba package was used to compile and parallelize each function before running. Simulations were run on the Cray CS300 supercomputer at the University of Hawaii at Manoa with `cpus-per-task` $= 20$ and `mem` $= 10,000$`MB`. All code (and this paper) can be found at the following Git repository:

https://github.com/glennga/micro-coa

In this implementation, the following details differ from the descriptions laid in the previous chapters:

1. The BFS we perform in subsection 3.3.2 is not guaranteed to explore nodes of the same depth in a deterministic manner, but still performs a BFS nonetheless. Numba parallelizes this process, which leads to this randomness.

2. In order to use algorithm 3.3.1, a common ancestor repeat length $\ell^{(1)}$ must be specified. *We assume $\ell^{(1)}$ to be a nuisance parameter.* Given the variance associated with this process, we feel it is safe to assume this parameter does not need to be as precisely defined as our other hyperparameters. We randomly select a repeat length from our observed data, and reduce the number of free parameters from four to three.

3. In an effort to save space, two independent vectors V and E are not allocated for use at the same time in algorithm 3.3.1. Instead we define vector V to hold the indices associated with each parent node to construct the edge set, and replace this with repeat lengths in the determination stage.

4. The order with which the probabilities of some likelihood θ are calculated are not guaranteed to be deterministic. Numba also parallelizes this process.

Symbol	Value / Type	Detail / Explanation
κ	3	Lower bound of our repeat length space $\mathbb{M}$.
Ω	30	Upper bound of our repeat length space $\mathbb{M}$.
n	100	Sample size, number of end individuals to simulate.
$\tau_N(e)$	$\exp\left({}^{N}/_{\binom{\rho(e)}{2}}\right)$	Generation determination function
N	$10,000$	Effective population size.
$\delta(e)$	$\delta_A(e)$	The distance function between two populations.
ϵ	0.55	ABC cutoff to define approximate.
T_1	100	Number of simulations to run to determine $\Pr(\mathcal{M}(\theta) \approx \mathcal{D}_i)$.
$\theta_c^{(1)}$	0.01	Starting c value for initial state of Markov chain $\theta^{(1)}$.
$\theta_d^{(1)}$	0.001	Starting d value for initial state of Markov chain $\theta^{(1)}$.
$g_c(c)$	$N(c, \sigma_c)$	Proposal function to generate new c values.
$g_d(d)$	$N(d, \sigma_d)$	Proposal function to generate new d values.
σ_c	0.001	Deviation associated with proposal function $g_c(c)$.
σ_d	0.001	Deviation associated with proposal function $g_d(d)$.
T_2	$55,000$	Running time of Metropolis sampler.
b_c	0.001	Bin size for c histogram.
b_d	0.0001	Bin size for d histogram.
$w_c(c)$	$\Gamma(\ldots)$	The assumed posterior distribution for c.
$w_d(d)$	$\Gamma(\ldots)$	The assumed posterior distribution for d.

Table 7.1: Table of our hyperparameters: their symbol, the value we chose, and a description of parameter itself.

5. In addition to recording the states required for Markov chain X in algorithm 6.3.1, we also record the time between acceptances (known as the *waiting time*), the likelihood of that point $\mathcal{L}(\theta^{(i)})$ (to avoid computing this later), the average distance associated with the generated data of the accepted parameter set and our observations, and the time of acceptance in terms of iterations.

7.2 Hyperparameters

Recall that we must specify hyperparameters in order to determine our parameters of interest: c, d. Each is listed in Table 7.1 with their symbol, the value we chose, and a brief description of hyperparameter itself. Below we give our reasoning for selecting each Table 7.1 value:

1. For all of the microsatellites we collected in chapter 5 from ALFRED, the lowest observed repeat length we found was $\ell = 3$. We set $\kappa = 3$ for this reason.

2. For all of the microsatellites we collected in chapter 5 from ALFRED, the highest observed repeat length we found was $\ell = 30$. We set $\Omega = 30$ for this reason.

3. The sample size n heavily influences the computation time of our simulator. Per Metropolis sampler iteration, we generate $n \cdot T_1 \cdot |\mathcal{D}|$ lineages. We found that $n = 100$

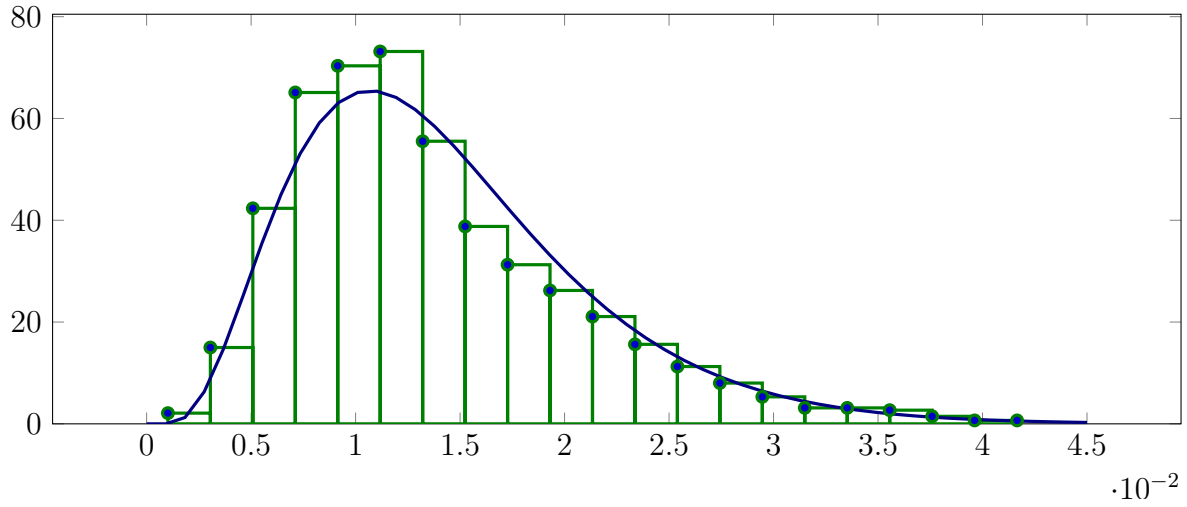

Figure 7.1: Frequency surface of c parameter. Our histogram was constructed using a bin size of $b_c = 0.001$ In blue the least-squares estimate of a Γ distribution for this histogram is displayed, with $a = 3.740, loc = 0.0009042, scale = 0.003572$.

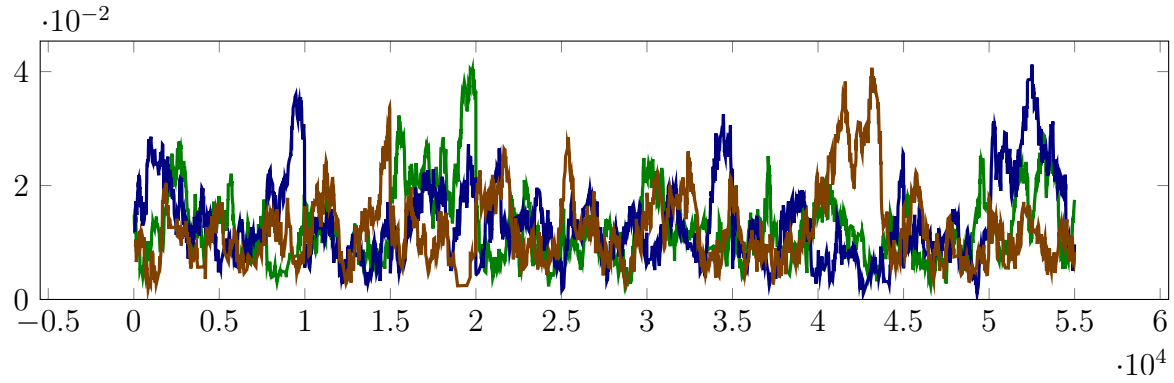

Figure 7.2: Trace plot of c for all three runs of the Metropolis sampler.

was on the lower end of sample sizes found in ALFRED observations, but this allowed us to simulate samples within a reasonable amount of time.

4. We chose to follow Hudson's approach and drew our edge lengths from an exponential distribution with the mean $N/\binom{\rho(e)}{2}$. We defined τ_N as a function which accepts an edge as input, determines the expected time to coalescence based on the working node's depth from the common ancestor node, and outputs a draw from this exponential distribution with this expected time as the mean.

5. We use a human effective population size of $N = 10,000$ individuals, taken from Takahata [24].

6. The decision to use the angular distance δ_A as our δ function was made arbitrarily. Future work includes looking into other such distance functions.

7. To determine ϵ, our starting parameter values $\theta_c^{(1)}, \theta_d^{(1)}$, and our proposal deviations

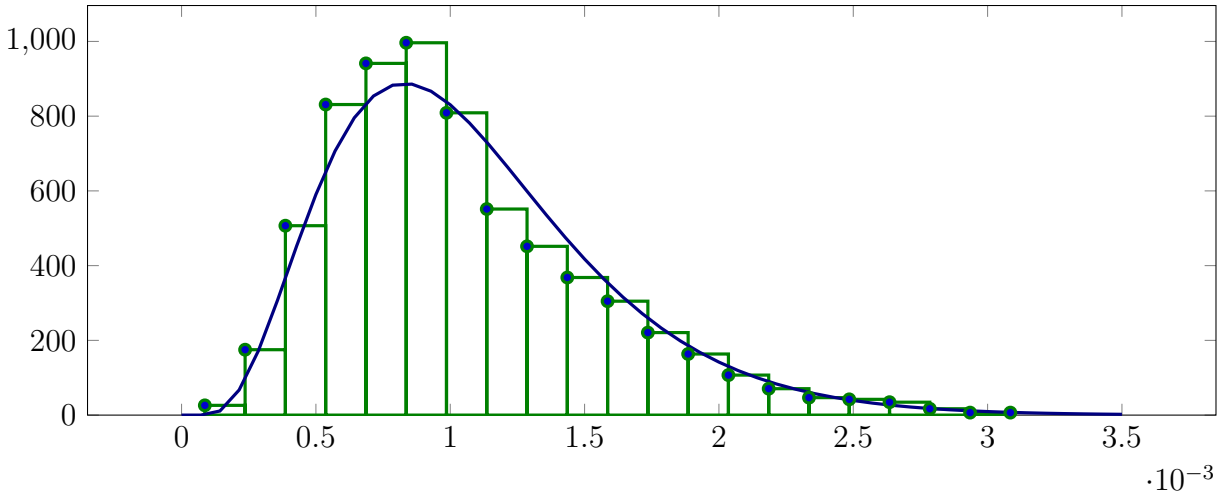

Figure 7.3: Frequency surface of d parameter. Our histogram was constructed using a bin size of $b_d = 0.0001$. In blue the least-squares estimate of a Γ distribution for this histogram is displayed, with $a = 4.000, loc = 7.203 - 05, scale = 0.0002524$.

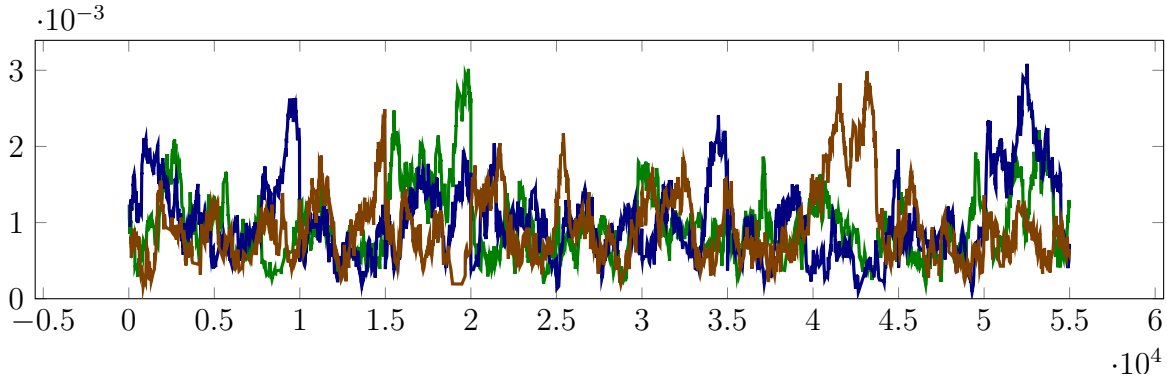

Figure 7.4: Trace plot of d for all three runs of the Metropolis sampler.

σ_c, σ_d, we started with some guess and refined these parameters as we went. ϵ itself depends on the other two parameters. We started at a high threshold $\epsilon \approx 0.9$, and decreased this number as we became more confident in our starting parameters over several instances of our Markov chains $\mathcal{X} = \{X_1, X_2, \ldots\}$. We found $\epsilon = 0.55$ to be a good compromise between accuracy and computation time.

8. As mentioned in the point for sample size, T_1 is another major factor in computation time. If T_1 is too small, we end up with probabilities that are not representative of our posterior. If T_1 is too large, our computation time becomes too long. We found that $T_1 = 100$ was able to reasonably account for our simulator variance.

9. The starting c parameter $\theta_c^{(1)}$ was tuned by running our sampler to obtain various Markov chains $\mathcal{X}$ and selecting the average of the most likely c values $\hat{c}$ for each chain. We rounded to the nearest 0.001, and obtained $\theta_c^{(1)} = 0.010$.

10. The starting d parameter was found in a similar fashion to the starting c parameter.

We rounded to the nearest 0.0001, and obtained $\theta_d^{(1)} = 0.001$.

11. We chose to sample from a normal distribution $g_c(c) = N(c, \sigma_c)$ centered at a given c for our proposal function c. This was the first symmetric distribution to come to mind, which is a requirement of the Metropolis sampler.

12. We also chose to sample from a normal distribution for parameter d: $g_d(d) = N(d, \sigma_d)$.

13. Associated with the proposal functions we defined in the previous two points are normal distribution deviations. A larger σ_c means that our sampler will on average generate proposals at greater distances from our current point than a sampler with a smaller σ_c. These were found in the same manner as our ϵ term: running several instances of sampler to obtain $\mathcal{X}$ and decreasing σ_c as we become more and more confident in our starting parameters. We found $\sigma_c = 0.0010$ after running this procedure.

14. Running a similar procedure for the proposal normal distribution deviation associated with d, we found $\sigma_d = 0.0001$.

15. Our Metropolis sampler is able to generate a Markov chain with $\Pr(\theta \mid \mathcal{D})$ as its equilibrium distribution *given an infinite amount of time*. To make this problem tractable, we must choose some T_2 such that our target distribution converged at some Metropolis iteration before T_2. We have selected $T_2 = 55,000$ for this run.

16. The histogram bin size is associated with the proposal function step size. A larger σ_c, σ_d will result in wider distributions, leading to an increase in b_c, b_d to ensure that our bin frequencies correctly resolve our posterior. This parameter was found after creating several different histograms with different b_c until some density appeared. We found $b_c = 0.001$ to resolve our posterior well given our current σ_c.

17. Using the same procedure for b_c, we found $b_d = 0.0001$ to resolve our posterior well given our current σ_d.

18. While creating several histograms for c, we observed that our density was positively skewed. Visually, we found that fitting our histogram to a gamma distribution aligned better than fitting our histogram to a normal distribution.

19. Again, while creating several histograms for d, we observed that our density was positively skewed. We went with a gamma distribution here as well.

7.3 Upward Constant Bias: c

In Figure 7.1, we describe the frequency of the c parameter. The data for the histogram comes from three independent runs of the Metropolis sampler, all with identical hyperparameters. The distribution below represents the curve of this frequency, with a bin size of $b_c = 0.001$ to construct the underlying histogram.

$$w_c(c) = \Gamma(a = 3.740, loc = 0.0009042, scale = 0.003572) \tag{7.1}$$

The most likely parameter $\hat{c}$ is given below:

$$\hat{c} = 0.01426 \pm 4.773 \cdot 10^{-5} \tag{7.2}$$

The trace plot for c, a plot of c vs. Metropolis sampler iterations is shown in Figure 7.2. A visual approach to assessing Markov chain convergence is to look at how our Markov chain moves as a function of time. All three are shown to move in the same neighborhood, although the ideal plot here would be a set of straight, thick, and tightly distributed lines. This indicates that we need to run our MCMC longer to strengthen our confidence that we are sampling from the correct distribution.

7.4 Downward Linear Bias: d

In Figure 7.3, we describe the frequency of the d parameter. The data for the histogram comes from three independent runs of the Metropolis sampler, all with identical hyperparameters. The distribution below represents the curve of this frequency, with a bin size of $b_d = 0.0001$ to construct the underlying histogram.

$$w_d(d) = \Gamma(a = 4.000, loc = 7.203 - 05, scale = 0.0002524) \tag{7.3}$$

The most likely parameter $\hat{d}$ is given below:

$$\hat{d} = 0.001082 \pm 2.549 \cdot 10^{-7} \tag{7.4}$$

Using both $\hat{c}$ and $\hat{d}$, we get the following focal unit for the Columbian populace:

$$\hat{\ell} \sim 13 \tag{7.5}$$

The trace plot for d is shown in Figure 7.4. Same with c, all three are shown to move in the same neighborhood, but we would need to run our MCMC longer to strengthen our confidence that our chain has converged.

Chapter 8

Future Work

In this chapter, we discuss the next steps associated with our research.

8.1 Evolutionary Simulator

The focus of this project was to find an appropriate evolutionary simulator and model for a single population. We believe that we have constructed a fast evolutionary simulator, but it could be more efficient. As of now, the running time of our evolutionary simulator is dependent on n and N. n must be large enough to account for the variance in this problem, but we can remove our reliance on N by drawing our mutations from some distribution instead of iterating from generation to generation. This increases the noise associated with our mutation model, but may make our simulator much faster.

8.2 Implementation Updates

All code for this project is current written in Python 3.7. We were able to get reasonable speedup using Numba to compile certain sections of our code (as opposed to only interpreting it), but it might be of interest to move to a compiled language like C++ altogether. It might also be of interest to start moving certain sections of the sampler (simulator, δ, etc...) to the GPU. This problem is highly parallelizable, but we are only running at most 20 tasks in parallel on the CPU. The GPU would allow us to run thousands of tasks in parallel, (ideally) resulting in less runtime.

8.3 Mutation Model

We were able to obtain estimates for our mutation model, but we need to further verify that these parameters are correct. There exists three main areas of future work:

1. Verifying the model using more populations. We were only able to use the Columbian populace here, but using the same methodology with other groups would give us different $\hat{c}$ and $\hat{d}$ values to compare to.

2. Comparing the total mutation rate to that of other `GATA` microsatellites in different literature.

3. Running our MCMC longer. As seen in the previous section, our trace was not the ideal shape but it showed hints toward convergence. The easiest solution here to increase our number of iterations T_2, which would show a straighter and thicker line if we have truly converged.

4. Retrieving more statistics about our Metropolis sampler. We visually assessed Markov chain convergence by looking at the trace plot, but there exist other metrics that could be explored (see [5]). This would allow us to tune our sampler with more precision, and possibly find better estimates for $\hat{c}$ and $\hat{d}$.

8.4 Demographic Models

Once we are confident in our mutation model, we can extend our work toward using the same methodology with *multiple* populations. The mutation model parameters then become hyperparameters to these demographic models as we try to estimate additional parameters such as population size, admixture, and divergence times. Below, we present several multi-population models from Stoneking and Krause that can be used to make various inferences [22].

1. Three population models: (Ancestors, African vs. Non African), (Ancestors, Modern Humans vs. Neanderthals), (Ancestors, Modern Humans vs. Denisovans). We can determine the population sizes of all populations and the time of each split.

2. A five population model: Ancestors, African One, African Two, Non African One, Non African Two. We can determine the admixture between the African and Non African populations, as well as the time associated with each split.

3. A seven population model: Ancestors, Modern Humans One, Modern Humans Two, Neanderthals One, Neanderthals Two, Denisovan One, Denisovan Two. Again, we can determine the admixture between each population as well as the time associated with each split.

Chapter 9

Bibliography

[1] R. Bellman. *Dynamic Programming*. Courier Corporation, Apr. 2013.

[2] J. M. Butler, M. D. Coble, and P. M. Vallone. STRs vs. SNPs: Thoughts on the future of forensic DNA testing. *Forensic Science, Medicine, and Pathology*, 3(3):200–205, Sept. 2007.

[3] M. C. Campbell and S. A. Tishkoff. The Evolution of Human Genetic and Phenotypic Variation in Africa. *Current Biology*, 20(4):R166–R173, Feb. 2010.

[4] S.-H. Cha. *Comprehensive Survey on Distance/Similarity Measures between Probability Density Functions*. 2007.

[5] M. K. Cowles and B. P. Carlin. Markov Chain Monte Carlo Convergence Diagnostics: A Comparative Review. *Journal of the American Statistical Association*, 91(434):883–904, 1996.

[6] A. Di Rienzo, A. C. Peterson, J. C. Garza, A. M. Valdes, M. Slatkin, and N. B. Freimer. Mutational processes of simple-sequence repeat loci in human populations. *Proceedings of the National Academy of Sciences of the United States of America*, 91(8):3166–3170, Apr. 1994.

[7] H. Ellegren. Heterogeneous mutation processes in human microsatellite DNA sequences. *Nature Genetics*, 24(4):400–402, Apr. 2000.

[8] H. Fan and J.-Y. Chu. A Brief Review of Short Tandem Repeat Mutation. *Genomics, Proteomics & Bioinformatics*, 5(1):7–14, 2007.

[9] J. C. Garza, M. Slatkin, and N. B. Freimer. Microsatellite allele frequencies in humans and chimpanzees, with implications for constraints on allele size. *Molecular Biology and Evolution*, 12(4):594–603, July 1995.

[10] R. Gemayel, J. Cho, S. Boeynaems, and K. J. Verstrepen. Beyond Junk-Variable Tandem Repeats as Facilitators of Rapid Evolution of Regulatory and Coding Sequences. *Genes*, 3(3):461–480, July 2012.

[11] I. C. Gray, D. A. Campbell, and N. K. Spurr. Single nucleotide polymorphisms as tools in human genetics. *Human Molecular Genetics*, 9(16):2403–2408, Oct. 2000.

[12] M. Hanada. Markov Chain Monte Carlo for Dummies. Aug. 2018.

[13] R. R. Hudson. Gene genealogies and the coalescent process. *Oxford Surveys in Evolutionary Biology*, 7:1–44, 1990.

[14] T. Jehan and S. Lakhanpaul. Single nucleotide polymorphism (SNP)–Methods and applications in plant genetics: A review. *IJBT Vol.5(4) [October 2006]*, Oct. 2006.

[15] J. Lintusaari, M. U. Gutmann, R. Dutta, S. Kaski, and J. Corander. Fundamentals and Recent Developments in Approximate Bayesian Computation. *Systematic Biology*, 66(1):e66–e82, Jan. 2017.

[16] P. Marjoram, J. Molitor, V. Plagnol, and S. Tavaré. Markov chain Monte Carlo without likelihoods. *Proceedings of the National Academy of Sciences*, 100(26):15324–15328, Dec. 2003.

[17] T. Ohta and M. Kimura. A model of mutation appropriate to estimate the number of electrophoretically detectable alleles in a finite population*. *Genetics Research*, 89(5-6):367–370, Dec. 2007.

[18] S. Rice. Review 4.18, Organic Evolution.

[19] O. Rose and D. Falush. A threshold size for microsatellite expansion. *Molecular Biology and Evolution*, 15(5):613–615, May 1998.

[20] R. Sainudiin, R. T. Durrett, C. F. Aquadro, and R. Nielsen. Microsatellite Mutation Models. *Genetics*, 168(1):383–395, Sept. 2004.

[21] S. Sankararaman, S. Mallick, N. Patterson, and D. Reich. The Combined Landscape of Denisovan and Neanderthal Ancestry in Present-Day Humans. *Current Biology*, 26(9):1241–1247, May 2016.

[22] M. Stoneking and J. Krause. Learning about human population history from ancient and modern genomes. *Nature Reviews Genetics*, 12(9):603–614, Sept. 2011.

[23] J. Sup Lee, M. G. Hanford, J. L. Genova, and R. Farber. Relative Stabilities of Dinucleotide and Tetranucleotide Repeats in Cultured Mammalian Cells. *Human Molecular Genetics*, 8(13):2567–2572, Dec. 1999.

[24] N. Takahata. Allelic genealogy and human evolution. *Molecular Biology and Evolution*, 10(1):2–22, Jan. 1993.

[25] A. F. Wright. Genetic Variation: Polymorphisms and Mutations. In *eLS*. American Cancer Society, 2005.

[26] X. Xu, M. Peng, Z. Fang, and X. Xu. The direction of microsatellite mutations is dependent upon allele length. *Nature Genetics*, 24(4):396–399, Apr. 2000.