

**FINAL REPORT OF THE  
UC DENVER MATHEMATICS CLINIC  
Modeling Mutation Rates for Avian Flu  
Neuraminidases**

Taught by  
Stephen C. Billups  
University of Colorado Denver  
Department of Mathematical Sciences  
P.O. Box 173364  
Denver, CO 80217-3364  
Stephen.Billups@cudenver.edu  
<http://www-math.cudenver.edu/~billups/>

Sponsored by  
Jack Horner, SAIC

**Participating students:**

Mahougnon Amanadi, Joseph Cavaleri, Angela Harris,  
Jeff Kenyon, Minjeong Kim, Zak Kirk, Marcela Kuzmiak,  
Jennifer Reinert, Shelley Speiss, Craig Tennenhouse,  
Michael Trujillo and Timothy Vis

Fall Semester 2007



# Contents

<b>Acknowledgements</b>	<b>vii</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Generative Models for Nucleotide Sequence Evolution</b>	<b>5</b>
2.1 Introduction . . . . .	5
2.2 Substitution Models . . . . .	7
2.2.1 Jukes Cantor Model . . . . .	9
2.2.2 Treegen: A Generative Implementation of Jukes Cantor	11
2.2.3 General Time Reversible Model . . . . .	11
2.3 Generating Sequences with Rose . . . . .	13
2.3.1 Description of the Algorithm . . . . .	14
2.3.2 Input Parameters . . . . .	15
2.4 Maximum Likelihood Estimation of Model Parameters with GARLI . . . . .	17
2.4.1 Description of the Algorithm . . . . .	17
2.4.2 Input Parameters . . . . .	18
2.4.3 Output Data . . . . .	20
2.5 Determining Rose Parameters from GARLI Output . . . . .	20
2.5.1 MATLAB Conversion Scripts . . . . .	21
2.6 Concluding Remarks . . . . .	21
2.A Rose Input File . . . . .	23
2.B GARLI Input File . . . . .	24
2.C GARLI Output File . . . . .	26
2.D MATLAB Conversion Functions . . . . .	29
2.D.1 gtr.m . . . . .	29
2.D.2 gouttrin.m . . . . .	30

2.D.3	garlitorose.m . . . . .	31
2.D.4	pairedist.m . . . . .	32
2.D.5	maketree.m . . . . .	32
2.D.6	rsequence.m . . . . .	35
2.D.7	gtrdef . . . . .	35
<b>3</b>	<b>Evaluating Phylogenetic Algorithms</b>	<b>37</b>
3.1	Introduction . . . . .	37
3.1.1	Literature Review . . . . .	38
3.2	Testing Environment and Data Set . . . . .	39
3.2.1	Hardware . . . . .	39
3.2.2	Generating the Data Set and Control Trees . . . . .	39
3.2.3	Sequencing the data sets with CLUSTALW . . . . .	41
3.2.4	Generating Phylogenetic Trees . . . . .	41
3.2.5	Comparing Trees . . . . .	43
3.3	Results . . . . .	45
3.4	Discussion . . . . .	45
3.4.1	Algorithms . . . . .	52
3.4.2	Behavior over Sequence and Set Size Increases . . . . .	53
3.5	Future Work . . . . .	54
3.5.1	Using Better Start Trees in GARLI . . . . .	54
3.5.2	Selection of Evolutionary Model . . . . .	55
3.5.3	Improved Experimental Design . . . . .	55
3.6	Conclusions . . . . .	55
3.A	Generating Sequence Data . . . . .	57
3.B	Using ROSE to Generate Sequences and Trees . . . . .	63
<b>4</b>	<b>The Effect of Tree Topology on Leaf Distances</b>	<b>65</b>
4.1	Introduction . . . . .	65
4.2	The Procedure . . . . .	66
4.3	Results . . . . .	68
4.4	Conclusion . . . . .	74
4.A	Project Details . . . . .	77
<b>5</b>	<b>Mathematical Model for the Mutation Rate of the Avian Flu Virus</b>	<b>83</b>
5.1	Introduction . . . . .	83
5.2	Background . . . . .	87



5.2.1	UPGMA - Unweighted Pair Group Method with Arithmetic Mean . . . . .	87
5.2.2	Neighbor Joining . . . . .	90
5.2.3	GARLI - Genetic Algorithm for Rapid Likelihood Inference . . . . .	96
5.3	Results . . . . .	101
5.3.1	UPGMA and Neighbor Joining Trees . . . . .	101
5.3.2	Maximum Likelihood Trees . . . . .	104
5.3.3	Parameters . . . . .	106
5.4	Conclusion and Future Work . . . . .	106
<b>6</b>	<b>Performing Multiple Sequence Alignment on Very Large Data Sets</b>	<b>109</b>
6.1	Introduction . . . . .	110
6.2	Multiple Sequence Alignment . . . . .	110
6.2.1	Programs for Multiple Sequence Alignment . . . . .	112
6.2.2	Comparing Multiple Sequence Alignments . . . . .	114
6.2.3	Literature Search . . . . .	115
6.3	Methods . . . . .	119
6.3.1	Computer Environment . . . . .	119
6.3.2	Generation of data sets . . . . .	119
6.3.3	Aligning and Scoring . . . . .	120
6.3.4	Generating Phylogenetic Trees . . . . .	120
6.4	Results . . . . .	120
6.5	Discussion . . . . .	121
6.5.1	Future Work . . . . .	127
6.6	Conclusions . . . . .	128
<b>7</b>	<b>Conclusions</b>	<b>131</b>
	<b>Bibliography</b>	<b>135</b>

# List of Figures

3.1	CLUSTALW MSA Generation Times (minutes)	42
3.2	BSD Example	44
3.3	Branch Score Distance, 100 sequences	46
3.4	Branch Score Distance, 500 sequences	46
3.5	Branch Score Distance, 1000 sequences	47
3.6	Branch Score Distance, 1500 sequences	47
3.7	Robinson Foulds Distance, 100 sequences	48
3.8	Robinson Foulds Distance, 500 sequences	48
3.9	Robinson Foulds Distance, 1000 sequences	49
3.10	Robinson Foulds Distance, 1500 sequences	49
3.11	Log Likelihood Scores	52
4.1	Analysis Flowchart	67
4.2	Accuracy of Pairwise Branch Distances (Fitch-Margoliash)	69
4.3	Accuracy of Pairwise Branch Distances (Neighbor Joining)	69
4.4	Precedence Metric, (Fitch-Margoliash)	70
4.5	Precedence Metric, (Neighbor Joining)	70
4.6	Distance Metrics (Fitch-Margoliash)	71
4.7	Distance Metrics (Neighbor Joining)	72
4.8	Precedence Distance vs. Topo Distance	73
4.9	RF Dist. vs. Number of Leaves	73
4.10	UPGMA Max/Mean	74
4.11	UPGMA Precedence	75
4.12	Distance Metrics, UPGMA	76
4.13	Newick Tree	77
5.1	Flowchart of the project.	85
5.2	GARLI flow chart.	97
5.3	Tree created by UPGMA.	102

5.4	Tree created with Neighbor Joining.	103
5.5	Tree created with UPGMA tree seed.	104
5.6	Tree created with Neighbor Joining tree seed.	105
6.1	MSA Generation Times (minutes)	121
6.2	Shift scores, 100 sequences	123
6.3	Shift scores, 500 sequences	123
6.4	Shift scores, 1000 sequences	124
6.5	Shift scores, 1500 sequences	124
6.6	RF Distance: Set Sizes 100, 500, 1000, 1500; Sequence Length 1000	125
6.7	Branch Score Distance: Set Sizes 100, 500, 1000, 1500; Sequence Length 1000	125
6.8	Log Likelihood Scores: Set Sizes 100, 500, 1000, 1500; Sequence Length 1000	126

# List of Tables

3.1	Tree generation times (minutes) for Neighbor Joining (NJ), UPGMAL, Maximum Likelihood (ML), GARLI . . . . .	50
3.2	Robinson Foulds (RF), Branch Score Distance (BSD), Log Likelihood (lnL) Scores . . . . .	51
3.3	GARLI generation times (minutes) and scoring, using Neighbor Joining (NJ) and Maximum Likelihood (ML) starting topologies . . . . .	56
3.4	GARLI scoring, using Maximum Likelihood (ML) starting topologies from Hasegawa (H) and Jukes-Cantor (JC) . . . . .	56
3.5	Maximum Likelihood (ML) scoring, using Hasegawa (H) and Jukes-Cantor (JC) Evolutionary Models . . . . .	56
6.1	Summary of MSA Programs (from Edgar, 2006 [10]) . . . . .	117
6.2	Typical Alignment Tasks (from Edgar, 2006 [10]) . . . . .	118
6.3	Scores for Generated MSAs . . . . .	122

## Acknowledgements

The success of the Mathematics Clinic program is largely dependent upon identifying an important and mathematically interesting problem, which will fuel the creative energies of our students. I am therefore deeply indebted to Jack Horner for giving us this project. It was just plain fun to work on, and forced us to learn a lot of math that you don't see in a traditional classroom. I am also, as always, grateful to the students who participated. Their enthusiasm, creativity and hard work inspired me and made this clinic a great experience. They are

Mahougnon Amanadi,  
Joseph Cavaleri,  
Angela Harris,  
Jeff Kenyon,  
Minjeong Kim,  
Zak Kirk,

Marcela Kuzmiak,  
Jennifer Reinert,  
Shelley Speiss,  
Craig Tennenhouse,  
Michael Trujillo,  
and Timothy Vis

Stephen C. Billups,  
Math Clinic instructor



# Chapter 1

## Introduction

This report summarizes the results of the Fall 2007 Mathematics Clinic. The Mathematics Clinic is a project-based course offered by the University of Colorado Denver Department of Mathematical Sciences. In the Mathematics Clinic, graduate and advanced undergraduate students work in teams to address a problem of interest to a sponsoring organization.

This semester, the project was sponsored by Jack Horner, from SAIC (Science Applications International). The project was motivated by concerns of a possible pandemic arising from a mutation of the H5N1 Avian Flu virus. Currently, this deadly virus cannot be transmitted from human to human. But there is a very real threat that the virus could mutate into a form that could be transmitted from human to human.

Currently, there are only two drugs available to treat the H5N1 virus. The most effective of these, oseltamivir (brand name Tamiflu), works by inhibiting the glyco-protein Neuraminidase. Within the world-wide population of the H5N1 virus, there is considerable variation of the composition of the Neuraminidase protein. And this variation influences the drug response. Thus, as the virus continues to evolve, there is risk that new mutations will render oseltamivir ineffective.

The math clinic was given the challenge of developing a mathematical model of the mutation rates of the gene coding for Neuraminidase, and then training this model using DNA sequences extracted from the Los Alamos National Laboratory Influenza Sequence Database [30]. Since the database does not record any ancestral relationships between the DNA sequences, there is no direct way to infer mutation rates. Instead, ancestral relationships must first be inferred by constructing a phylogenetic tree. Once the phylogenetic

tree is constructed, a maximum likelihood procedure can be used to tune the parameters of the mutation model so that they agree with the data as accurately as possible.

Within the context of this general framework, several questions needed to be addressed. As such, students in the clinic were divided into four teams, each with a different focus as described below:

**Team 1 (Mahougnon Amanadi, Joseph Cavaleri, and Timothy Vis):** This team was responsible for investigating mathematical models for DNA mutation. This team started by trying to develop their own probabilistic model from scratch. In parallel to this effort, they also studied the literature on nucleotide substitution models. The main result of their work was to recommend a General Time Reversible Model [29] as the working model for this clinic. The results of this project are documented in Chapter 2.

**Team 2 (Jeff Kenyon and Zak Kirk):** This team was one of two teams investigating phylogenetic algorithms. In their study, they compared the accuracy and computational efficiency of various algorithms on simulated data. Of particular interest was the question of how accurately does each algorithm reproduce the original tree topology (which was used to create the data)? This project is the subject of Chapter 3.

**Team 3 (Angela harris, Graig Tennenhouse and Michael Trujillo):** Another question relative to phylogenetic trees is “how important is it to correctly identify the true tree topology?” The significance of this question lies in the fact that the most computationally expensive phylogenetic algorithms spend most of their time exploring different tree topologies. While cheaper algorithms may not get the topology correct, they may still do a good job in determining reasonable phylogenetic distances between sequences. To investigate this question, this team of students performed a study to understand the sensitivity of the accuracy of pairwise distances to the accuracy of the tree topology. Their results are given in Chapter 4

**Team 4 (Jennifer Reinert, Shelley Speiss, Minjeong Kim and Marcela Kuzmiak):** The final team of students was responsible for using the LANL influenza sequence database to fit the model developed by Team 1. Their work, and a discussion of the final model are described in Chapter 5



In addition to these four teams, one student (Jeff Kenyon), did a separate honors project in which he investigated various algorithms for performing multiple sequence alignment (as potential improvements over the industry standard CLUSTALW algorithm). This project is relevant to the clinic project since the first step in determining a phylogenetic tree is to perform a multiple sequence alignment. As such, Jeff's honors project is included as Chapter 6.



# Chapter 2

## Generative Models for Nucleotide Sequence Evolution

By Mahougnon Amanadi, Joseph Cavaleri and Timothy Vis

### Abstract

We describe a simple model predicting the evolution of nucleotide sequences through nucleotide substitution. The model generates a phylogenetic tree with variable branch lengths as well as the sequences at each node. The underlying assumptions of this model are that it must be time-reversible and that the probability of a substitution occurring must increase with the branch length. In addition, we describe algorithms for using maximum likelihood estimation to determine the parameters of this model from a given set of sequences.

### 2.1 Introduction

In recent years, the avian flu has sparked fears of a devastating epidemic reminiscent of the 1918 epidemic that killed approximately 50 million people. Given the pathogenic strength of the H5N1 avian flu virus, the potential for disaster if it gains the capability to be transmitted directly from one human to another is great. As yet, no known form of the H5N1 avian flu virus is communicable from one human to another; however, the possibility that a future mutant might gain this capability is certainly present and certainly

calls for appropriate preparations. In its current forms, the H5N1 virus can be effectively combated with the drug oseltamivir. This drug interferes with the function of the neuraminidase protein, which allows the virus, once it has reproduced, to break free from a host cell to invade further host cells. In order to be properly prepared to fight any future variants of this virus, it is important to consider how effective oseltamivir will be in its interference with their neuraminidase. Preliminary work due to Jack Horner indicates that the effectiveness of oseltamivir on the neuraminidase of distinct forms of the H5N1 virus is closely linked to the phylogenetic distances between the nucleotide sequences for the neuraminidase of each variant and the sequence for a reference individual [20]. This suggests that a knowledge of how the neuraminidase sequences of future variants of the avian flu might look will be invaluable in determining how effective oseltamivir will be on these mutants, as well as how urgent the need for new developments in the fight against this virus is.

In order to predict the future neuraminidase sequences for the H5N1 avian flu virus, a generative model for mutations is necessary. Such a model must have the capability of being trained to fit the existing data for the neuraminidase sequences already known, and recorded in the database hosted by the Los Alamos National Laboratories [30]. The output of the model desired should consist of a phylogenetic tree for future mutations of the neuraminidase sequences, along with predicted future sequences. Training this model involves the inference of a phylogenetic tree to describe the ancestry of the extant sequences.

The construction of phylogenetic trees has been an active area of research for many years; however, most of the work that has been done has been concentrated not on extrapolating from present species or sequences to future species or sequences, but on inferring the ancestry of groups of existing species or sequences. Fortunately, however, many of the same principles used to infer ancestral phylogenetic trees can be used to generate possible future members of a population. The same models which are used to determine how the extant sequences were evolved from a common ancestry can be used to guess at how the extant sequences will continue to evolve.

Evolution of a sequence where reproduction is entirely asexual is described purely by mutations. These mutations can be described most simply as taking three forms: substitutions, insertions, and deletions. In a substitution, one nucleotide is replaced by another nucleotide in the same position, while the length of the sequence remains constant. In an insertion, a string is

inserted into some part of the nucleotide sequence, lengthening that sequence. In a deletion, a string is deleted from some part of the nucleotide sequence, shortening that sequence.

Unfortunately, there is no established consensus on how insertions and deletions work [47]. While different techniques exist and are used in multiple sequence alignment algorithms, there is no consensus on the relationship between evolutionary distances and the insertion and deletion structures of mutations, nor any corresponding model for the simulation of insertions and deletions.

On the other hand, a great deal of work has been done in modeling nucleotide substitution, and a number of widely accepted models of varying complexity exist. In what follows, we include descriptions of several models of nucleotide substitution.

Our purpose has been to research and describe models for mutation in order to provide a mechanism for the prediction of future nucleotide sequences for the neuraminidase gene in the avian flu. To that end, we have utilized the publicly available software packages GARLI [59], described in Section 2.4 and Rose [47], described in Section 2.3 to implement the general time reversible model of substitution, described in Section 2.2.3, and have created our own software, utilizing MATLAB, to interpret and convert the output from GARLI into a usable input file for Rose. In the following sections, we first of all describe in some detail the substitution models we studied. We then discuss how Rose can be used to generate sequences, including in our discussion an overview of the appropriate parameter settings and a description of the program. Following this, we explore how GARLI may be utilized to determine the appropriate parameters for mutation, again including recommendations on parameter settings to ensure consistency between the programs and the chosen general time reversible model. We describe our MATLAB scripts for interpreting and converting the output from GARLI into the input for Rose. Finally, we include as appendices sample input files for Rose and GARLI, a sample output file from GARLI, and the code from our MATLAB programs.

## 2.2 Substitution Models

In the most general sense, as described by Rodríguez et al [42], the process of nucleotide substitution can be considered as a stochastic process, described

by a  $4 \times 4$  matrix  $P(t)$  that varies over time, where  $P(t)_{i,j}$  represents the probability that nucleotide  $i$  has mutated to nucleotide  $j$  after a period of time  $t$ .

$$P = \begin{pmatrix} 1 - \sum f_{AX}(t) & f_{AC}(t) & f_{AG}(t) & f_{AT}(t) \\ f_{CA}(t) & 1 - \sum f_{CX}(t) & f_{CG}(t) & f_{CT}(t) \\ f_{GA}(t) & f_{GC}(t) & 1 - \sum f_{GX}(t) & f_{GT}(t) \\ f_{TA}(t) & f_{TC}(t) & f_{TG}(t) & 1 - \sum f_{TX}(t) \end{pmatrix}$$

Here each  $f_{XY}(t)$  is the probability function that a nucleotide with symbol  $X$  mutates to one with symbol  $Y$  in a time  $t$ , and the summation is simply over the remaining functions in the given row of the matrix. Notice that each row then sums to one, as desired. In practice, a mutation after a period of time  $t$  is determined probabilistically at each nucleotide site. A nucleotide with symbol  $A$ , for example, would have a probability of  $f_{AC}(t)$  of mutating to symbol  $C$ , a probability of  $f_{AG}(t)$  of mutating to symbol  $G$ , and a probability of  $f_{AT}(t)$  of mutating to symbol  $T$ .

The determination of  $P(t)$  is by no means trivial, and, in general,  $P(t)$  cannot be determined analytically [59], but only approximated with numerical methods. As a result, the substitutions are more commonly described by means of a matrix giving the rates of each type of substitution relative to the proportion of that nucleotide, a rate that is assumed to be constant over time. This gives the matrix  $Q$ , which is independent of time, with rows summing to zero. The diagonal entries are determined by simply adding the negations of the remaining entries in a row.

$$Q = \begin{pmatrix} -\sum r_{AX} & r_{AC} & r_{AG} & r_{AT} \\ r_{CA} & -\sum r_{CX} & r_{CG} & r_{CT} \\ r_{GA} & r_{GC} & -\sum r_{GX} & r_{GT} \\ r_{TA} & r_{TC} & r_{TG} & -\sum r_{TX} \end{pmatrix}$$

Although we do not discuss the mathematics of this relation (which is a consequence of certain differential equations),  $P$  and  $Q$  are related by the equation

$$P = e^{Qt}.$$

We next restrict our discussion to some situations in which the substitution rates are related. In particular, we examine the case where time is reversible—that is, where the rate of substitution from character  $X$  to character  $Y$  is the same as the rate of substitution from character  $Y$  to character

$X$ . In such a case, the root of a phylogenetic tree constructed under the model cannot be deterministically located, and any root location is equally likely. We discuss two particular models: the Jukes-Cantor model [22] and the General Time Reversible model [29].

### 2.2.1 Jukes Cantor Model

The Jukes Cantor model, introduced by Jukes and Cantor in 1969 [22], is the simplest substitution model possible. In this model, the rate of substitution from one nucleotide to another is assumed to be independent of the actual nucleotides being substituted. In other words, the matrix  $Q$  is given as follows:

$$Q = \begin{pmatrix} -3\alpha & \alpha & \alpha & \alpha \\ \alpha & -3\alpha & \alpha & \alpha \\ \alpha & \alpha & -3\alpha & \alpha \\ \alpha & \alpha & \alpha & -3\alpha \end{pmatrix}$$

For this model, it is possible, and relatively simple, to determine  $P$  from  $Q$ . Consider the probability that a given nucleotide has symbol  $X$  at time  $t + dt$ . There are two cases to be considered. If this nucleotide had symbol  $X$  at time  $t$ , the probability that it will continue to have symbol  $X$  at time  $t + dt$  is  $1 - 3\alpha dt$ . If this nucleotide had symbol  $Y$  at time  $t$ , the probability that it will mutate to symbol  $X$  at time  $t + dt$  is  $\alpha dt$ . We obtain the following equation.

$$P_{YX}(t + dt) - P_{YX}(t) = \alpha(1 - P_{YX}(t))dt - 3\alpha P_{YX}(t)dt$$

This simplifies to the differential equation

$$\begin{aligned} P'_{YX}(t) &= \alpha(1 - P_{YX}(t)) - 3\alpha P_{YX}(t) \\ &= \alpha(1 - 4P_{YX}(t)) \end{aligned}$$

which has the solution

$$P_{YX} = \frac{1}{4} - \frac{1}{4}Ae^{-4\alpha t}.$$

To determine  $A$ , we consider two possible initial conditions:  $Y = X$  and  $Y \neq X$ , or, alternately,  $P_{YX}(0) = 1$  and  $P_{YX}(0) = 0$ . In the first case, solving for  $A$ , we obtain

$$P_{XX} = \frac{1}{4} + \frac{3}{4}e^{-4\alpha t}.$$

In the second case, again solving for  $A$ , we obtain

$$P_{YX} = \frac{1}{4} - \frac{1}{4}e^{-4\alpha t}.$$

Thus, the matrix  $P$  is determined as

$$P = \begin{pmatrix} \frac{1}{4} + \frac{3}{4}e^{-4\alpha t} & \frac{1}{4} - \frac{1}{4}e^{-4\alpha t} & \frac{1}{4} - \frac{1}{4}e^{-4\alpha t} & \frac{1}{4} - \frac{1}{4}e^{-4\alpha t} \\ \frac{1}{4} - \frac{1}{4}e^{-4\alpha t} & \frac{1}{4} + \frac{3}{4}e^{-4\alpha t} & \frac{1}{4} - \frac{1}{4}e^{-4\alpha t} & \frac{1}{4} - \frac{1}{4}e^{-4\alpha t} \\ \frac{1}{4} - \frac{1}{4}e^{-4\alpha t} & \frac{1}{4} - \frac{1}{4}e^{-4\alpha t} & \frac{1}{4} + \frac{3}{4}e^{-4\alpha t} & \frac{1}{4} - \frac{1}{4}e^{-4\alpha t} \\ \frac{1}{4} - \frac{1}{4}e^{-4\alpha t} & \frac{1}{4} - \frac{1}{4}e^{-4\alpha t} & \frac{1}{4} - \frac{1}{4}e^{-4\alpha t} & \frac{1}{4} + \frac{3}{4}e^{-4\alpha t} \end{pmatrix}$$

It should be noted that in this situation, as noted above,  $P$  can also be determined as a matrix exponential:

$$P = e^{Qt};$$

however, we will not discuss the mathematical justification of this statement. In general, it is this method of determining  $P$  that is most useful, particularly when more general and more complicated models are used.

It is common practice to measure time in such a way that the mean number of substitutions at any given site in a nucleotide sequence in a unit of time is one. This assumption leads to an  $\alpha$  value of  $\frac{1}{3}$ , and the matrix can be adjusted accordingly.

If we increase time to infinity in each case, we observe the following:

$$\begin{aligned} \lim_{t \rightarrow \infty} \left( \frac{1}{4} + \frac{3}{4}e^{-4\alpha t} \right) &= \frac{1}{4} \\ \lim_{t \rightarrow \infty} \left( \frac{1}{4} - \frac{1}{4}e^{-4\alpha t} \right) &= \frac{1}{4} \end{aligned}$$

so that the probabilities of observing any particular nucleotide become equal and independent of the original nucleotide present in a given position.

While the Jukes Cantor model of substitution is easy to work with and is useful for illustrating some of the mathematics that underlie models of evolution, it suffers from some serious defects. The most glaring defect is that the instantaneous rates of mutation are assumed to be equal. This is generally not a valid assumption. In general, the rates may differ significantly. Additionally, the equilibrium frequencies of the bases are assumed to be identical. Once again, this is an unrealistic assumption.



On the other hand, the Jukes Cantor model is time-reversible. The probability of a nucleotide with symbol  $X$  mutating to one with symbol  $Y$  is precisely the same as that of one with symbol  $Y$  mutating to one with symbol  $X$ . The general time reversible model, which we discuss in the following section, remedies both of the aforementioned defects, while maintaining time-reversibility.

### 2.2.2 Treegen: A Generative Implementation of Jukes Cantor

In order to understand how data might be generated under the Jukes Cantor model, we constructed a program, Treegen in MATLAB. This program creates a phylogenetic tree whose branch lengths are determined by a gamma distributed random variable for a given number of generations and outputs a MATLAB phytree object, along with DNA sequences for each node of the tree. The gamma distribution is a two-parameter continuous probability distribution with the following probability density function:

$$f(x) = x^{\alpha-1} \frac{e^{-\frac{x}{\beta}}}{\beta^{\alpha} \Gamma(\alpha)} \quad x, \alpha, \beta > 0.$$

This program takes five parameters, these being the two parameters of the gamma distribution, a rate parameter for the Jukes Cantor model (scaling for the number of mutations per unit time), a root DNA sequence, and a number of generations.

Using these parameters, Treegen determines the branch lengths using the gamma distribution (chosen with little justification), and then mutates the sequence according to the Jukes Cantor model, with the given parameter. Having no reasonable justification for the gamma distribution of branch lengths, and having discovered Rose, we did not further develop Treegen, preferring to work with the existing and established software available. The files necessary for executing Treegen are available on request.

### 2.2.3 General Time Reversible Model

Following the Jukes Cantor substitution model, several further models were introduced, which brought various improvements to the simplicity in this model. The Kimura 2-parameter model [24] introduced a second substitution

rate, differentiating between transitions (between members of the same base pair) and transversions (between members of opposite base pairs).

In 1981, Felsenstein [13] used a model that accounted for differing nucleotide base frequencies. Aside from maintaining the base frequencies, this model assumes equal substitution rates in the same manner as the Jukes-Cantor model.

Four years later, the same process was applied to the Kimura 2-parameter model by Hasegawa, Kishino, and Yano [17] combining the improvements of both Kimura and Felsenstein. Further models extended the number of rates to three, allowing for differing transition rates with one transversion rate [25], or for differing transversion rates with one transition rate [49].

We concern ourselves with the most general of these models, attributed to Lanave et al. [29]. This model allows for variable base frequencies and includes six differing rates of substitution, one for each pair of distinct base frequencies. As such, this model consists of a total of ten parameters: four parameters ( $\pi_A, \pi_C, \pi_G, \pi_T$ ) for the base frequencies of the four nucleotides, and six parameters ( $r_{AC}, r_{AG}, r_{AT}, r_{CG}, r_{CT}, r_{GT}$ ) for the rates of substitution between these six bases. In general, however, only eight of these parameters are considered free. The fourth base frequency is determined by the necessity that the four base frequencies sum to one, while the six rates are typically given as relative rates [5], with the sixth rate set to one. Thus, we obtain as the parameters a vector of base frequencies

$$\pi = (\pi_A \quad \pi_C \quad \pi_G \quad 1 - (\pi_A + \pi_C + \pi_G))$$

and a relative rate matrix

$$R = \begin{pmatrix} & R_{AC} & R_{AG} & R_{AT} \\ R_{AC} & & R_{CG} & R_{CT} \\ R_{AG} & R_{CG} & & 1 \\ R_{AT} & R_{CT} & 1 & \end{pmatrix}.$$

These parameters can be used to determine  $Q$ . Since the rates of substitution in  $Q$  are given relative to the proportion of each nucleotide, we multiply each entry in the matrix by the base frequency of the corresponding column to ensure that the appropriate base frequencies are maintained, and that the rate of substitutions are indeed symmetric, in that the absolute rate of substitutions from  $X$  to  $Y$  is identical to the absolute rate of substitutions from  $Y$  to  $X$ . We further multiply each entry in the matrix by a constant  $\mu$

which scales the matrix for the appropriate mean instantaneous substitution rate.

$$Q = \begin{pmatrix} -\sum_{j \neq 1} Q_{1,j} & \mu\pi_C R_{AC} & \mu\pi_G R_{AG} & \mu\pi_T R_{AT} \\ \mu\pi_A R_{AC} & -\sum_{j \neq 2} Q_{2,j} & \mu\pi_G R_{CG} & \mu\pi_T R_{CT} \\ \mu\pi_A R_{AG} & \mu\pi_C R_{CG} & -\sum_{j \neq 3} Q_{3,j} & \mu\pi_T \\ \mu\pi_A R_{AT} & \mu\pi_C R_{CT} & \mu\pi_G & -\sum_{j \neq 4} Q_{4,j} \end{pmatrix}$$

In order to obtain the appropriate mean substitution rate of one, we consider the total quantity of substitutions and divide by this quantity. The proportion of sites having base  $X$  is  $\pi_X$ . The proportion of these that will mutate to  $Y$  is given by  $\mu\pi_Y R_{XY}$ , giving a total of  $\mu\pi_X\pi_Y R_{XY}$  of these substitutions per site. Thus, the total number of expected substitutions per site, which is one, yields the following equation [5].

$$1 = 2\mu \sum_{X \neq Y} \pi_X \pi_Y R_{XY},$$

so that

$$\mu = \frac{1}{2 \sum_{X \neq Y} \pi_X \pi_Y R_{XY}}.$$

and therefore,

$$Q_{i,j} = \frac{\pi_j R_{ij}}{2 \sum_{X \neq Y} \pi_X \pi_Y R_{XY}} \quad i \neq j$$

$$Q_{i,i} = -\sum_{j \neq i} Q_{i,j}.$$

Given the form for  $Q$ , it is possible to use numerical methods to then approximate the appropriate probability matrix  $P(t)$  when desired.

## 2.3 Generating Sequences with Rose

Having discussed the prevalent models for nucleotide substitution, we now turn our attention to a means of generating actual sequence data that fits these models. We pay particular attention to the software package Rose [47], developed for this very purpose. Rose is a very versatile package, whose applications stretch far beyond our usage; however, we have, for various

reasons, chosen not to apply much of the power of Rose. The material that follows is adapted from [47] as well as the program manual, both of which are readily and freely available on-line at the following web address: <http://bibiserv.techfak.uni-bielefeld.de/rose>.

### 2.3.1 Description of the Algorithm

Rose begins with a root sequence, which may either be specified by the user or randomly generated based on a vector of character frequencies input by the user. Rose also requires a mutation guide tree. This may, again, be specified by the user. If no tree is user-specified, Rose generates a uniform binary tree with 1023 nodes (depth 9), adjusting the edge lengths so that the average distance along a shortest path through the tree between two nodes is equal to a user-defined value.

When the tree is constructed, Rose uniformly chooses the desired number of nodes, choosing either from the full set of nodes or from the leaf nodes (as set by the user), and prunes unnecessary branches from the tree before constructing the sequences.

Once these structures are in place, Rose mutates sequences to create the descendants of each sequence. Rose conducts each of the three varieties of mutation at every step. To perform substitutions, Rose requires a substitution rate matrix whose  $(i, j)$  entry represents the probability of  $i$  replacing  $j$  in one unit time. This matrix is simply the transpose of  $P(t)$  as discussed earlier, when  $t = 1$ . Rose adjusts this matrix for the branch length in the appropriate manner to be exactly equivalent to  $P(t)$  (where  $t$  is the branch length) in its action on the characters of the nucleotide sequence.

Rose allows for substitution probabilities to be scaled linearly at each site by a user-defined vector. This allows for the specification of sites at which substitutions cannot occur, occur less frequently, or occur more frequently than the supplied substitution rate.

After the substitutions are performed, subsequences are inserted into and deleted from the child sequence according to user-defined insertion and deletion probability functions. Since no standard models exist for the insertion and deletion (which Stoye, Evers, and Meyer acknowledge), we will not discuss this mechanism, except to note its existence and potential for further research.

Finally, Rose outputs the desired number of sequences, together with the true mutation guide tree and the true multiple sequence alignment.

### 2.3.2 Input Parameters

The input for Rose consists of a text file, with each of several lines specifying certain parameters used by the program. Rose requires each of the following inputs:

Name	Type	Description
TheAlphabet	String	Alphabet of characters to be used in sequences
TheFreq	Floating point vector	Vector indicating equilibrium frequency of each character in the sequences
TheInsFunc	Floating point vector	Vector indicating probabilities for lengths of insertions
TheDelFunc	Floating point vector	Vector indicating probabilities for lengths of deletions
and one or the other of the following		
ThePAMMatrix	Floating point matrix	Matrix of substitution probabilities
TheDNAModel	String	Specification of a standard substitution model "JC", "HKY", "F81", "F84", or "K2P"

A number of other input lines may be included. We list those that are the most important here and refer the reader to the manual (included in the software package) for the remainder.

Name	Type	Description
SequenceLen	Integer	Average sequence length
SequenceNum	Integer	Number of sequences to be reported
ChooseFromLeaves	Boolean	Output sequences from all nodes or just from leaves
TheTree	PHYLIP Tree	The mutation guide tree
Relatedness	Integer	Average distance between sequences
TheSequence	String	Root sequence
TheMutationProbability	Floating point vector	Site specific substitution rate adjustment factor

We include an input file for Rose with the appropriate settings as an appendix to this document (Appendix 2.A). However, some of the settings bear further explanation in terms of the choices made. The setting “TheAlphabet” will naturally be set to “ACGT”, as we are concerned with DNA sequences. The setting “TheFreq” cannot be arbitrarily set, but must be determined from the given data upon which the evolution modeled by Rose is being trained. We return to this in Section 2.4. As we are not concerning our discussion with insertions and deletions, “TheInsFunc” and “TheDelFunc” will both be set to all zero vectors so that no insertions and deletions can occur. Finally, as we prefer to use the general time reversible model, discussed in Section 2.2.3, we enter the appropriate matrix, again determined from the data upon which our evolution model is trained, as “ThePAMMatrix” and do not use the parameter “TheDNAModel”.

Since we do not include insertions and deletions, every sequence created will have the same length. Thus, we need not concern ourselves with “SequenceLen”. To maximize our control, we provide a uniform binary tree on 1023 nodes, where we specify the branch lengths using the technique described by [47]. As such, we have the choice of 1023 nodes from which to choose our sequences. In order to gain the true effect of a random sampling of available sequences, we recommend that the number of reported sequences be kept relatively small, for instance “SequenceNum” being at most 100. In order to reflect the true reality, it seems that choosing only from leaf nodes of a uniform binary tree will not produce a sample of temporally spaced data, as desired. As such, we set “ChooseFromLeaves” to “false” so that all nodes of the tree are available as output sequences. This leads to a tree that need not be ultra-metric (where by ultra-metric we mean that all leaf nodes have the same distance from the root), a restriction that generally does not fit the known data.

“TheSequence” will simply be a sequence from the data available. It seems that to get an accurate picture, several iterations of Rose with several different sequences from the known data will be necessary.

Finally, “TheMutationProbability” will not be used. Although a large body of work supports site specificity in various forms as producing more accurate phylogenetic trees [59], obtaining enough information to determine the exact nature of the site specificity seems intractable at this point. The most common method of describing rate heterogeneity appears to be the use of a gamma-distributed random variable with mean one and shape parameter  $\alpha$  which adjusts the substitution rates at each site. Introduced by Yang

[56, 57], such a technique improves the inferred phylogeny.

Unfortunately, while the shape parameter can be readily estimated, determining the exact multipliers to use at each point in the sequence is difficult. In order to maintain accuracy, it would be important to maintain not only the same distribution of rate heterogeneity multipliers, but also the particular multipliers at each position in the sequence. Failure to do so could easily result in sequences with lower substitution rates than the mean being assigned a higher substitution rate than the mean in the generation of further sequences. Thus, in order to avoid this as a problem, we set “TheMutationProbability” to be a vector consisting of all ones and evaluate the other parameters under the assumption that no rate heterogeneity exists.

We turn next to the methods used in evaluating the parameters used by Rose.

## 2.4 Maximum Likelihood Estimation of Model Parameters with GARLI

In the preceding sections, we discussed several parameters of Rose that needed to be determined from known data. In this section, we discuss the software package GARLI [59], a program which uses maximum likelihood techniques to infer phylogenies. The use of this program will allow us to infer the desired parameters for Rose. Much of the information that follows is derived from [59], as well as from the manual included in the distribution of GARLI. GARLI is freely available on-line at the following web address: <http://www.bio.utexas.edu/faculty/antisense/garli/Garli.html>.

### 2.4.1 Description of the Algorithm

GARLI uses a genetic algorithm approach to find and estimate the phylogenetic tree and model parameters generating a given set of pre-aligned sequences. The program takes as its input a set of aligned nucleotide or amide sequences, and, optionally, a collection of potential phylogenetic trees for the data. Where large numbers of sequences are used, seeding GARLI with trees determined by techniques such as Neighbor Joining or UPGMA is highly recommended to both minimize the runtime and optimize the output.

If no trees are provided, GARLI randomly creates a number of trees with one leaf node for each sequence in the alignment. GARLI estimates

parameters for the appropriate model of evolution (selected by the user) and then proceeds to iterate the genetic algorithm.

At each generation of the genetic algorithm, GARLI possesses a specified number of possible phylogenetic trees, together with their model parameters. The likelihood score of each tree is calculated, and certain optimizations of model parameters and branch lengths is carried out. Following this procedure, the tree with the highest final likelihood score is carried to the next generation. There, mutations of the topology, branch lengths, and evolutionary model parameters are used to fill out the population of that generation. This process repeats until certain stopping conditions (selected by the user) are met. Generally, these stopping conditions consist of a maximum number of total generations, or a maximum number of generations without any significant change to the likelihood score of the best tree or the topology of the best tree.

After GARLI has completed its genetic algorithm, it conducts an extensive branch length optimization to obtain the best possible tree. This tree, together with the final model parameters constitute the main part of the output of GARLI.

## **2.4.2 Input Parameters**

As with Rose, the input for GARLI consists of a text file, specifying various parameters for the program. The following inputs will be significant.



Name	Options	Description
datafname	filename	Name of the file containing the non-interleaved sequences
streefname	filename	Name of the file containing the initial tree and model parameters
outputphylptree	0 or 1	Determines whether a phylip file will be output
ratematrix	1rate, 2rate, 6rate, fixed	Determines the substitution model being used
statefrequencies	equal, empirical, estimate, fixed	Sets how the equilibrium base frequencies of the nucleotides are determined
invariantsites	none, estimate, fixed	Sets the method of determining how many sites have no substitution
ratehetmodel	none, gamma, gammafixed	Sets the assumed model of rate heterogeneity over sites

The remaining parameters in GARLI are not important to our purposes, and we refer the reader to the GARLI manual. These parameters deal with how GARLI logs its progress, what the exact stopping conditions are, and how the genetic algorithm operates.

Again, we include an input file for GARLI with the appropriate settings as an appendix (Appendix 2.B). As discussed earlier in Section 2.3.2, we use the most general time-reversible model of evolution. This may correspond to either the “6rate” or “fixed” option for the “ratematrix” parameter. Since, however, one of our purposes in using GARLI is to determine parameters such as the rate matrix, using the “fixed” option, which requires the rate matrix as a part of the input, is unreasonable. Therefore, we use the “6rate” option for “ratematrix”.

Our use of Rose also requires that we input the base frequencies, and we again allow GARLI to estimate these for us. The option “estimate” allows the most freedom in determining the base frequencies, while “empirical” takes the base frequencies present in the input data. We use “estimate” for the “statefrequencies” parameter in order to account for both the ancestral and extant sequences.

As we earlier stated, we do not assume any rate heterogeneity whatsoever. This automatically requires that “invariantsites” be set to “none” and “ratehetmodel” be set to “none”. It should be mentioned that this last setting

requires that we set “numratecats” to “1”, although we do not discuss this parameter.

Finally, we set “outputphylptree” to “1” to output the phylip file, which can more readily be used to determine the parameters for Rose.

### 2.4.3 Output Data

The output from GARLI consists of several files. Most of these files log the progress of the genetic algorithm and the time taken by GARLI, and we do not discuss these. The file of interest is given the tag `.best.tree`, and it is with this file that we continue our discussion. We attach a sample GARLI output file as an appendix (Appendix 2.C).

This file contains several elements:

1. A legend numbering the sequences
2. The optimized phylogenetic tree in Newick format, and
3. The model parameters, including:
  - (a) “rmat”, the five non-trivial entries in the relative rate matrix  $R$
  - (b) “base”, the three free entries in the base frequency vector
  - (c) “rates”, the rate heterogeneity model, and
  - (d) “pinv”, the proportion of invariant sites.

Under the input parameters specified, “pinv” will be zero and “rates” will be “equal”. The elements of this file in which we are interested are “rmat” and “base”. It is these elements which will be used to determine the appropriate inputs for Rose.

## 2.5 Determining Rose Parameters from GARLI Output

The elements of “base” give the base frequencies, in order, of A, C, and G. Since the frequencies must sum to one, the frequency of T is easily calculated by summing the other frequencies and subtracting from one. This determines the fourth element of the vector which becomes the parameter “TheFreq” in Rose.

We determined in Section 2.2.3 the means of converting from the relative rate matrix  $R$  to the probability matrix  $P$ , and noted in Section 2.3.2 that the rate matrix used by Rose was exactly the transpose of  $P$ . Thus, determining  $P$  and taking the transpose gives us the desired matrix to fulfill the “ThePAMMatrix” in Rose.

For the input “TheTree”, we create a uniform binary tree with 1023 nodes. The branch length is determined by the average distance between all nodes (which we find using the tree output by GARLI) and dividing by 14 ( $2(7 - 2)$ ), where 7 is the depth of the tree, as described in [47]).

We have created a collection of MATLAB functions to perform the appropriate conversions from the GARLI output to the Rose configuration file.

### 2.5.1 MATLAB Conversion Scripts

The MATLAB functions we have created take only a single input, that being the raw filename of the output files for GARLI (without any type extensions). These functions create the appropriate Rose input stream file, having the same base filename, so that a given combined run of GARLI and Rose is unified under a single filename. The basic MATLAB function is titled `gtr.m`. This function takes the filename. It then calls `gouttrin.m` to extract the relative rate matrix and the base frequency vector from the GARLI output file. Using `garlitorose.m`, the relative rate matrix is converted to the probability matrix used by Rose. `gtr.m` converts this matrix into the proper string format for Rose, and does the same process for the base frequency vector. The function, `pairedist.m` determines the average distance  $d_{av}$  needed to construct the tree. This distance is used to calculate the appropriate branch length for a uniform binary tree with 2047 nodes (1024 leaves), which is passed to `maketree.m`, which constructs such a tree. Using this data, `gtr.m` creates the input file for Rose. Parameters that are not altered are included by way of a separate defaults file, which is included in the input file created. These scripts are included in Appendix 2.D.

## 2.6 Concluding Remarks

Using the complementary capabilities of Rose and GARLI, we are able to use the General Time Reversible model of nucleotide substitution to model the evolution of DNA sequences over time. We use GARLI to determine

the appropriate parameters that fit the existing and known data, our own scripts to reformat the parameters for Rose, and Rose to generate potential future representatives. This presents a useful tool for estimating the evolutionary relationships, and, in particular, phylogenetic distances, between flu specimens now present, and flu specimens which may appear in the future.

Naturally, this suggests a number of avenues of further research. We have repeatedly mentioned the lack of an accepted model for insertions and deletions, and it would be of great interest to see an appropriate model developed, or even to study any existing models to account for the effects of insertions and deletions on the future population.

Additionally, we have not explored the effect of site-specific rate heterogeneity. Exploration of this might indicate that the desired relationships do not depend on the exact positions that differ, but only on the actual distribution of the rate factors. This could also be an interesting avenue of research.

Finally, it might be of interest to study the effect of fixing various model parameters. If this did not result in significant changes, it might save a significant amount of time and computing power.

## 2.A Rose Input File

```
StdOut = False;
OutputFilename = "roseoutput";
OutputFilebase = "roseoutput";
SequenceSuffix = ".fas";
AlignmentFormat = "PHYLIP";
AlignmentWithAncestors = False;
AlignmentSuffix = ".phy";
TreeSuffix = ".tre";
SequenceOutputLen = 60;
SeedVal = None
SequenceNum = 10;
InputType = 4;
Relatedness = 1;
ChooseFromLeaves = False;
TreeWithSequences = True;
TreeSequencesWithGaps = False;
TreeWithAncestors = False;
TheTree = None;
TheSequence = "ACTCTCGCTAAATCGGTT";
ThePAMMatrix = [[0.97,0.01,0.01,0.01],[0.01,0.97,0.01,0.01],
                [0.01,0.01,0.97,0.01],[0.01,0.01,0.01,0.97]];
TheAlphabet = "ACGT";
TheFreq = [0.25,0.25,0.25,0.25];
TheInsertThreshold = 0;
TheDeleteThreshold = 0;
TheMutationProbability = [1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
                          1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0];
TheDNAModel = None;
MeanSubstitution = None;
TransitionBias = None;
TTratio = None;
NumberOfRuns = 1;
TheInsFunc = [0];
TheDelFunc = [0];
```

## 2.B GARLI Input File

```
[general]
datafname = rana.phy
constraintfile = none
streefname = ranastart.tre
ofprefix = ranaGarli
randseed = -1
availablememory = 512
logevery = 10
saveevery = 100
refinestart = 1
outputeachbettertopology = 1
enforcetermconditions = 1
genthreshfortopoterm = 10000
scorethreshforterm = 0.05
significanttopochange = 0.01
outputphyliptree = 1
outputmostlyuselessfiles = 0
writecheckpoints = 0
restart = 0

ratematrix = 6rate
statefrequencies = estimate
ratehetmodel = none
numratecats = 1
invariantsites = none

[master]
nindivs = 4
holdover = 1
selectionintensity = .5
holdoverpenalty = 0
stopgen = 5000000
stoptime = 5000000

startoptprec = .5
minoptprec = .01
```

```
numberofprecreductions = 20
treerejectionthreshold = 50.0
topoweight = 1.0
modweight = .05
brlenweight = 0.2
randnniweight = 0.1
randsprweight = 0.3
limsprweight = 0.6
intervallength = 100
intervalstostore = 5

limsprrange = 6
meanbrlenmuts = 5
gammashapebrlen = 1000
gammashapemodel = 1000
uniqueswapbias = 0.1
distanceswapbias = 1.0

bootstrapreps = 0
inferinternalstateprobs = 0
```

## 2.C GARLI Output File

```
#nexus

begin trees;
translate
  1  temporariaDMH84R1,
  2  boylliMVZ148929,
  3  luteiventris_MT_MVZ191016,
  4  luteiventris_WA_MVZ225749,
  5  muscosaMVZ149006,
  6  auroraMVZ13957,
  7  cascadaeMVZ148946,
  8  sylvaticaMVZ137426,
  9  sylvaticaDMH84R43,
  10 septentrionalesDCC3588,
  11 grylioMVZ175945,
  12 virgatipesMVZ175944,
  13 okaloosae,
  14 clamitansJSF1118,
  15 heckscheriMVZ164908,
  16 catesbianaX12841,
  17 catesbianaDMH84R2,
  18 maculataKU195258,
  19 vibicariaMVZ11035,
  20 warszewitshiiJSF1127,
  21 palmipesVenAMNHA118801,
  22 palmipesEcuKU204425,
  23 bwanaQCAZ13964,
  24 Sp_1_ecuadorQCAZ13219,
  25 vaillantiKU195299,
  26 julianiTNHC60324,
  27 sierramadrensisKU195181,
  28 psilonotaKU195119,
  29 tarahumaraeKU194596,
  30 zweifeliJAC7514,
  31 pustulosaJAC10555,
  32 pipiensJSF1119,
```



33 pipiensY10945,  
34 dunnijSF1017,  
35 montezumaeJAC8836,  
36 sp\_2\_mex\_JSF1106,  
37 chiricahuensisJSF1063,  
38 chiricahuensisJSF1092,  
39 subaquavocalis,  
40 palustrisJSF1110,  
41 areolataJSF1111,  
42 sevosaUSC8236,  
43 capitoSLU003,  
44 spectabilisJAC8622,  
45 forreriJSF1065,  
46 tlalociJSF1083,  
47 berlandieriJSF1136,  
48 neovolcanicaJSF960,  
49 blairiJSF830,  
50 omiltemanaJAC7413,  
51 magnaocularisJSF1073,  
52 yavapaiensisJSF1085,  
53 sp\_7\_JaliscoJSF1000,  
54 macroglossaJAC10472,  
55 macroglossaJSF7933,  
56 taylori286,  
57 sp\_4\_Panama,  
58 sp\_5\_CostaRichDMH86\_210,  
59 sp\_6\_CostaRicaDMH86\_225,  
60 sp\_8\_PueblaJAC9467,  
61 oncaLVT3542,  
62 sp\_3\_MichoacanJSF955,  
63 sphenoccephalaUSC7448,  
64 utriculariaJSF845;

tree best = [&U][-24577.556][ r 2.456458 6.982543 3.1091439  
0.78076136 16.206955 b 0.32764061 0.22446358 0.18008002  
0.26781579 ]((59:0.02404639,(58:0.02070131,57:0.01334301):  
0.00446231):0.00419777,((54:0.00840710,55:0.00535950):  
0.01125188,56:0.03899525):0.00973353,(((44:0.03095155,50:  
0.04124302):0.00424475,62:0.02285186):0.00286198,

```

((((33:0.00173383,32:0.00611292):0.01873047,((36:0.01530141,
(39:0.00646480,37:0.01675962):0.00103116,38:0.00691203):
0.00787128):0.00549242,(34:0.00866646,35:0.01376239):
0.01247487):0.01718679):0.02347588,((27:0.06669873,((29:
0.02632106,31:0.05330202):0.01080682,(30:0.03935032,28:
0.05162246):0.01301134):0.02706716):0.01936555,((((24:
0.01358757,(21:0.06073455,22:0.00448249):0.01918471):0
.00785586,23:0.02173856):0.01336076,(26:0.04113013,25:
0.05159345):0.01287414):0.03760387,((19:0.04187329,20:
0.06035407):0.04059833,18:0.04725818):0.01485366):0.01430515,
(((10:0.02104576,((14:0.00458409,13:0.00431589):0.00813538,
(15:0.02206284,(16:0.00000001,17:0.00103954):0.01855267):
0.00319522):0.00549825,(12:0.03330980,11:0.04140663):
0.00502277):0.00735102):0.03034461,(8:0.00314856,9:0.00207632):
0.04017379):0.01312487,(((4:0.00050339,3:0.00364522):
0.02192013,(2:0.03693488,1:0.05728056):0.01158852):0.00577521,
((6:0.01261546,7:0.01448233):0.01253667,5:0.01819152):
0.01491158):0.03772422):0.02398099):0.00982403):0.05148397):
0.00758177,(40:0.01764721,(41:0.02638977,(42:0.00213685,43:
0.00794626):0.01777335):0.00645975):0.01048866):0.02196754,
((64:0.00852435,63:0.01256110):0.01415808,(49:0.00599877,
(47:0.00454896,(48:0.00120913,46:0.00244213):0.00260805):
0.00522494):0.01682433):0.00329553):0.00385245):0.00539220,
((45:0.03757620,(51:0.05529845,53:0.03859883):0.01303302):
0.00431893,((52:0.00493604,61:0.00604112):0.01862597,60:
0.04305593):0.00565256):0.00460141):0.00473571);
end;
begin paup;
clear;
gett file=ranaGarli.best.tre storebr;
lset userbr nst=6 rmat=(2.45645803 6.98254297
3.10914390 0.78076136 16.20695498) base=(0.32764061
0.22446358 0.18008002) rates=equal pinv= 0.00000000;
end;

```

## 2.D MATLAB Conversion Functions

### 2.D.1 gtr.m

```
function gtr=gtr(string)

%The input is a string giving the filebase name for the
%output files from a run of GARLI. This function will
%search the GARLI output files and create a suitable input
%file for Rose.

output=string;
trefile=sprintf('%s.best.tre',output);
phyfile=sprintf('%s.best.tre.phy',output);
[rmat, base]=gouttrin(trefile);

%Here we obtain the proper probability matrix and format
%it as a string in the format Rose takes as input.

pmat=garlitorose(base,rmat);
pmats1=mat2str(pmat);
pmats2=strrep(pmats1,' ','[[ ');
pmats3=strrep(pmats2,'] ',' ]');
pmats4=strrep(pmats3,',' ', ');
pmats=strrep(pmats4,' ',' ');

%Here we obtain the base frequency vector and format
%it as a string in the format Rose takes as input.

fbase=[base,1-(base(1)+base(2)+base(3))];
fbases1=mat2str(fbase);
fbases=strrep(fbases1,' ',' ');

%Here we use the Phylip tree output by GARLI to construct
%a uniform binary tree with which to seed Rose. This tree
%has the property that the mean distance between two nodes
%is the same as that in the GARLI output tree. The tree has
%2047 nodes from which the sequences will be chosen.
```

```

dav=pairdist(phyfile);
b=dav*0.0625;
tree=maketree(b);

%Here we obtain a sequence from the LANL database with which
%to seed Rose. The sequence is randomly chosen from those in
%the database.

sequences=rsequence;

%We now combine the appropriate elements into an input file
%for Rose.

fid=fopen(output,'w');
fprintf(fid, '%include gtrdef\n\n');
fprintf(fid, strcat('OutputFilebase = "', string, '"\n'));
fprintf(fid, strcat('ThePAMMatrix = ', pmats, '\n'));
fprintf(fid, strcat('TheFreq = ', fbases, '\n'));
fprintf(fid, strcat('TheTree = ', tree, '\n'));
fprintf(fid, strcat('TheSequence = "', sequences, '"\n'));
begin{verbatim}

```

## 2.D.2 gouttrin.m

```

function [rmat, base] = gouttrin(file)
%
%   gouttrin.m  function to output the rmat and base vectors
%               from a garli output.
%
%   INPUT      file      is a string containing the file name
%                   of the garli output
%
%   OUTPUT     rmat      relative rate vector
%               base     base frequencies
%
%   AUTHOR     Joseph Cavaleri

```

```

%
%-----
%
fid=fopen(file);
S=fscanf(fid, '%c');

br=strfind(S, 'rmat');
bb=strfind(S, 'base');
be=strfind(S, 'rates');
rmats=S((br+6):(bb-3));
bases=S((bb+6):(be-3));
rmat=str2num(rmats);
base=str2num(bases);
return

```

### 2.D.3 garlitorose.m

```

%This function should take the GARLI output files and
%use it to adjust the Rose input file appropriately.
%Thus, it should pull the appropriate GARLI output
%parameters (the base frequency vector and the rate
%vector) to pass the appropriate rate matrix and base
%frequencies to Rose, and should determine the average
%distance between nodes of the output tree to pass to
%Rose as well.

```

```

function garlitorose = garlitorose(freq, rate)

```

```

piA=freq(1);
piC=freq(2);
piG=freq(3);
piT=1-(freq(1)+freq(2)+freq(3));

```

```

Rac=rate(1);
Rag=rate(2);
Rat=rate(3);
Rcg=rate(4);

```

```

Rct=rate(5);
Rgt=1;

total=piA*piC*Rac+piA*piG*Rag+piA*piT*Rat+piC*piA*Rac
      +piC*piG*Rcg+piC*piT*Rct+piG*piA*Rag+piG*piC*Rcg
      +piG*piT*Rgt+piT*piA*Rat+piT*piC*Rct+piT*piG*Rgt;

Q=[-(piC*Rac+piG*Rag+piT*Rat),piC*Rac,piG*Rag,piT*Rat;
   piA*Rac,-(piA*Rac+piG*Rcg+piT*Rct),piG*Rcg,piT*Rct;
   piA*Rag,piC*Rcg,-(piA*Rag+piC*Rcg+piT*Rgt),piT*Rgt;
   piA*Rat,piC*Rct,piG*Rgt,-(piA*Rat+piC*Rct+piG*Rgt)]*1/total;

PP=expm(Q);

P=PP';

garlitorose=P;
end

```

#### 2.D.4 pairdist.m

```

function dav = pairdist(string)

tree=phytreeread(string);

D=pdist(tree,'nodes','all','squareform',true);

sz=size(D);

ns=ones([sz(1) 1]);

dav=ns'*D*ns/(sz(1)*sz(1));
end

```

#### 2.D.5 maketree.m

```

function maketree=maketree(branch);

```

```

branches=num2str(branch);
string='';
for x=0:1023
    if mod(x,2)==0
        if mod(x,1024)==0
            string=strcat(string,'((((((((((a',num2str(x),
                ':', branches, ', '));
        elseif mod(x,512)==0
            string=strcat(string,'((((((((((a',num2str(x),
                ':', branches, ', '));
        elseif mod(x,256)==0
            string=strcat(string,'((((((((((a',num2str(x),
                ':', branches, ', '));
        elseif mod(x,128)==0
            string=strcat(string,'((((((((((a',num2str(x),
                ':', branches, ', '));
        elseif mod(x,64)==0
            string=strcat(string,'((((((((((a',num2str(x),
                ':', branches, ', '));
        elseif mod(x,32)==0
            string=strcat(string,'((((((((((a',num2str(x),
                ':', branches, ', '));
        elseif mod(x,16)==0
            string=strcat(string,'((((((((((a',num2str(x),
                ':', branches, ', '));
        elseif mod(x,8)==0
            string=strcat(string,'((((((((((a',num2str(x),
                ':', branches, ', '));
        elseif mod(x,4)==0
            string=strcat(string,'((((((((((a',num2str(x),
                ':', branches, ', '));
        else
            string=strcat(string,'(a',num2str(x),
                ':', branches, ', '));
        end
    else
        if mod(x+1,1024)==0;
            string=strcat(string, 'a', num2str(x), ', ',

```

```

        branches, '):', branches, '):', branches, '):',
        branches, '):', branches, '):', branches, '):',
        branches, '):', branches, '):', branches, '):',
        branches, ')');
elseif mod(x+1,512)==0
    string=strcat(string, 'a', num2str(x), ':',
        branches, '):', branches, '):', branches, '):',
        branches, '):', branches, '):', branches, '):',
        branches, '):', branches, '):', branches, '):',
        branches, ')');
elseif mod(x+1,256)==0
    string=strcat(string, 'a', num2str(x), ':',
        branches, '):', branches, '):', branches, '):',
        branches, '):', branches, '):', branches, '):',
        branches, '):', branches, '):', branches, ')');
elseif mod(x+1,128)==0
    string=strcat(string, 'a', num2str(x), ':',
        branches, '):', branches, '):', branches, '):',
        branches, '):', branches, '):', branches, '):',
        branches, '):', branches, ')');
elseif mod(x+1,64)==0
    string=strcat(string, 'a', num2str(x), ':',
        branches, '):', branches, '):', branches, '):',
        branches, '):', branches, '):', branches, '):',
        branches, ')');
elseif mod(x+1,32)==0
    string=strcat(string, 'a', num2str(x), ':',
        branches, '):', branches, '):', branches, '):',
        branches, '):', branches, '):', branches, ')');
elseif mod(x+1,16)==0
    string=strcat(string, 'a', num2str(x), ':',
        branches, '):', branches, '):', branches, '):',
        branches, '):', branches, ')');
elseif mod(x+1,8)==0
    string=strcat(string, 'a', num2str(x), ':',
        branches, '):', branches, '):', branches, '):',
        branches, ')');
elseif mod(x+1,4)==0

```



```

        string=strcat(string, 'a', num2str(x), ':',
            branches, '):', branches, '):', branches, ',');
    else
        string=strcat(string, 'a', num2str(x), ':',
            branches, '):', branches, ',');
    end
end
end
maketree=string;

```

### 2.D.6 rsequence.m

```

function rsequence = rsequence()

seqs = fastaread('trimmedSequenceSet.fas');
n=1102;
f=ceil(n.*rand);
rsequence = seqs(f).Sequence;

```

### 2.D.7 gtrdef

```

# default Rose parameters for GTR

StdOut = False
SequenceSuffix = ".fas"
AlignmentFormat = "PHYLIP"
AlignmentSuffix = ".phy"
TreeSuffix = ".tre"
SequenceOutputLen = 60
SequenceNum = 100
ChooseFromLeaves = False
TreeWithAncestors = True
TheAlphabet = "ACGT"
TheInsertThreshold = 0.00
TheDeleteThreshold = 0.00
TheInsFunc = [0.]
TheDelFunc = [0.]

```



# Chapter 3

## Evaluating Phylogenetic Algorithms

By Jeff Kenyon and Zak Kirk

### Abstract

This report evaluates several of the most popular algorithms for the generation of phylogenetic trees, in terms of their speed and accuracy. This comparison is made through the use of generated random sequences in which the mutation rate and the structure of the evolutionary tree is known *a priori*. Through the evaluation of sixteen experimental sets, we conclude that GARLI is superior to neighbor joining, UPGMA, or a “pure” maximum likelihood method at minimizing the log likelihood score, a metric taking both topology and branch distance into account.

### 3.1 Introduction

The problem to be investigated in this sub-project was to determine which of several popular approaches to phylogenetic tree generation produced the best tree. This is of interest in the current Math Clinic, since the quality of the phylogenetic tree would have an impact on the mutation rate derived from the tree.

This problem included a significant sub-problem, namely, how could the relative “goodness” of a set of phylogenetic trees produced from the LANL

data set be quantified? A quick search of the literature did not reveal a solution, and therefore, our group determined that our investigation should use generated data sets, where the correct tree structure would be known. We could then attempt to reconstruct the phylogenetic tree using multiple approaches, and compare the resulting trees to the known, correct tree. We used the publicly available ROSE software [46] to generate the data sets.

The project involved a further sub-problem of identifying the metrics in use for comparing phylogenetic trees. We identified three scoring metrics in common use: Robinson-Foulds (RF) score, Branch Score Distance (BSD), and Quartets Distance. We adopted the RF and BSD metrics, but were unable to find publicly available software for Quartets Distance. We also tracked Log Likelihood (lnL) scores for Maximum Likelihood (ML) and GARLI trees.

We concluded from our experimental findings that GARLI was the best program of those tested for minimizing the lnL score (a metric taking both tree topology and branch distance into account). The primary drawback of GARLI is its long running time; however, further experimentation revealed that running time can be significantly reduced by providing as starting topology the tree generated from a “pure” ML approach.

### 3.1.1 Literature Review

We were unable to find broad studies offering comprehensive evaluations of the algorithmic or heuristic approaches to phylogenetic tree constructions. The available studies indicated that ML was equal to or superior to the other methods with which it was compared.

Kuhner and Felsenstein [27] evaluated parsimony, compatibility, maximum likelihood, Fitch-Margoliash, and Neighbor-Joining methods. It found ML to be the best method overall, although for short sequence lengths, the distance-matrix methods could sometimes generate better results. It also showed that, when evolutionary rates varied, all the tested methods had problems.

Russo, *et. al.* [43] evaluated the distance methods of neighbor joining and minimum evolution, as well as parsimony and ML, using both amino acid and nucleotide sequences. They concluded that the major factor in constructing a correct tree was not the algorithm used, but the number of amino acids or nucleotides used in the analysis: in their testing, the genes with 377 or more codons led to the correct tree, while genes with 312 or fewer codons did not.

Hollich, *et. al.* [19] compare the standard Neighbor Joining algorithm

with several variants (as implemented by the programs BIONJ, FastME, and Weighbor). Their results were that BIONJ was the best of the lot, while FastME performed poorly on long branches, and Weighbor was considerably slower than the others.

## 3.2 Testing Environment and Data Set

### 3.2.1 Hardware

All sequencing and tree generation activities were performed on a Lenovo T60 laptop, with dual T2400 1.83GHz processors, and 2GB of RAM. The operating system was Windows XP Professional (Service Pack 2), and except where noted, all processing took place under Windows XP.

It should be noted that execution times for various activities should be used as a guideline only. The computer was not dedicated to the task, and other CPU-intensive tasks of varying duration may have executed during the course of, for example, a single GARLI run. In short, the situation was analogous to what would be encountered on any other multi-user machine.

### 3.2.2 Generating the Data Set and Control Trees

Two data sets were generated: the first was a kind of “homebrew” approach, utilizing Perl [55] to create files of FASTA sequences, using randomizations and naïve assumptions, while the second approach utilized a more conventional approach to data generation.

The Perl program for generating sequence data used an *e. coli* nucleotide strand as basic sequence. The mutation rate was  $10^{-4}$  (one mutation per 1000 nucleotides), and the mutation events could be, with equal probability, mutations, insertions, or deletions, the latter two taking place in groups of three nucleotides. The maximum offspring per individual per generation was set at five, but the actual number was a random integer up to the maximum. FASTA sequence output formatting was provided through the BioPerl package.

The Perl program used the following algorithm:

1. To generate a set of sequences of a given length  $n$ , read in the first  $n$  characters of the *e. coli* sequence, and declare this the root sequence;

2. If the set is not yet of the required size, create a new generation by creating offspring for each sequence in the previous generation:
  - (a) Determine the number of mutations per offspring. The number of mutations is a selection from a random distribution, where the mean is the sequence length multiplied by  $10^{-4}$ , and the standard deviation is the mean multiplied by .1;
  - (b) Determine the number of offspring. The number of offspring is a selection from a random distribution, with a minimum of one and a maximum of five;
  - (c) For each offspring, carry out the required number of mutations:
    - i. Choose randomly whether the mutation is to be a swap, and insertion, or a deletion;
    - ii. If a swap, choose a position at random and exchange the current nucleotide for one of the remaining three;
    - iii. If an insertion, make up a sequence of three nucleotides at random, and insert it at a random position within the sequence;
    - iv. If a deletion, select a position at random, adjust position so that the deletion takes place on an amino acid boundary, and delete the next three characters.
  - (d) Make up the necessary FASTA header information for the sequence.
3. Repeat until the total number of sequences exceeds the number needed.
4. Remove excess sequences from the last generation until the set has the exact number needed.
5. Output the sequences in FASTA format.

While allowing the project team to get up and running in evaluating tree generation algorithms quickly, it became apparent that the assumptions made in the data generation were inconsistent with the tree generation algorithms. Using these data sets, neighbor joining proved to be the superior technique, easily beating out ML and GARLI on a consistent basis, a result which was contradicted by the literature (see Section 3.1.1, above). Since these results would have required significant additional research to be defensible, we chose to find another approach to data generation.

Through the literature review process, the ROSE program [46] for generating sequence data and the associated tree was found. ROSE uses a probabilistic model, easily adjustable through configuration files, to generate DNA sequences using one of several possible evolutionary models. In the process of generating sequence data, the program creates the true phylogenetic tree (our “control trees”), and the correct sequence alignment. Notes on the project’s use of ROSE and the configuration input file are shown in Section 3.B.

The ROSE software is intended primarily for Unix platforms. For this project, an executable was built from downloaded source code on the cygwin platform [18], a Unix-like environment that runs on top of Windows.

### 3.2.3 Sequencing the data sets with CLUSTALW

Using a multiple sequence alignment program to align the sequence set is a necessary prerequisite to generating a phylogenetic tree.

ROSE produces the correct sequence alignment for each generated set. However, since the goal of this project was to determine which tree algorithm did the best job of reconstructing the tree, and since the LANL set of sequences was to be aligned using CLUSTALW, we decided to use CLUSTALW alignments for our experiment. Therefore, CLUSTALW alignments were created for the 16 generated data sets. Generation times are shown in Figure 3.1.

The details of CLUSTALW are beyond the scope of this project. Interested readers are referred to [50].

### 3.2.4 Generating Phylogenetic Trees

Neighbor Joining (NJ) and UPGMA trees were obtained using the DNADIST program (to generate the distance matrix) and the NEIGHBOR program, both in the PHYLIP [11] (version 3.67) suite of tools.

ML trees were generated using the PHYML [16] program (version 2.4.4). The program offers several substitution models for DNA, however, for this project (except where noted) the default Hasegawa model was used.

(It should be noted that the DNAML program within PHYLIP is also capable of generating ML trees. However, using it in this project proved problematic, as it was quite slow, and hung on a number of the “homebrew” data sets. It was not used for ROSE-generated data sets, although it is conceivable that it behaves better on data conforming to a more traditional evolutionary model.)

Set Size	Length 50	Length 100	Length 500	Length 1000
100	0.00938	0.03	0.67167	2.63333
500	0.19833	0.675	14.48333	57.46667
1000	0.83166	2.55	55.88333	225
1500	1.83333	5.86667	125	495

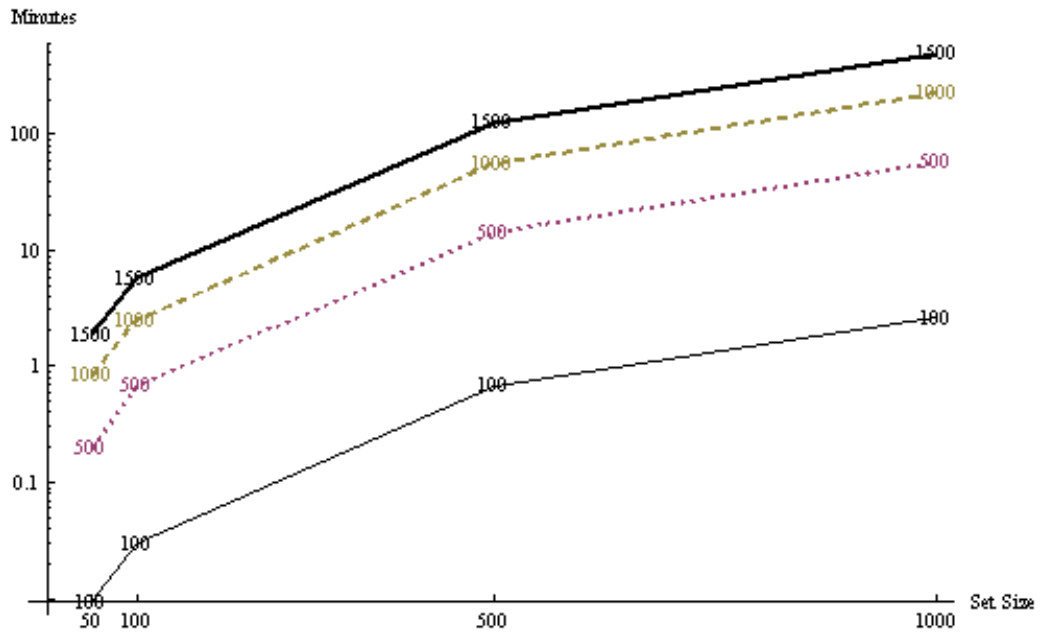


Figure 3.1: CLUSTALW MSA Generation Times (minutes)



GARLI [59], version 0.95, was used to find the best ML tree using a genetic algorithm approach. Using this software was understood to be mandatory, since one of the goals of the project was to determine whether GARLI's longer running times resulted in "better" phylogenetic trees.

Details of the tree generation algorithms are beyond the scope of this paper, but are available in the Background section of the full report.

### 3.2.5 Comparing Trees

In testing phylogenetic tree construction programs, it was obviously necessary to find a way of determining a measure of the accuracy of the trees produced [48]. In looking for this metric we considered the fact that our original program for generating synthetic data (a semi-probabilistic mutating of an existing genetic sequence) produced a tree without branch length. We decided to use the Robinson-Foulds (RF) distance [41], which is widely used in the domain. The RF distance has the advantage of not needing branch length in its calculation, but is instead solely on the topology of the trees being compared.

Other widely used distance measures for phylogenetic trees are the Branch Score Distance (BSD) [27] and the Quartets distance. After reviewing the RF scores we decided to also look at the BSD distance to try to explain some anomalous data.

For the BSD, the branch lengths are the metric used to determine the "closeness" of the trees. It is calculated by summing the squared differences between the branch lengths. In the case of tree  $T_1$  having a branch that is not present in the other tree,  $T_2$ , the tree  $T_2$  is given a corresponding branch with length 0. Thus, if both trees have branches  $\{\{A, D\}, \{B, C, E\}\}$ , the sum contains the square of the difference between the branch lengths. If one tree has the branch and the other does not, it contains the square of the difference between the branch length and zero (in other words, the square of that branch length). The BSD takes this sum of squared differences and computes its square root. Using the tree in Figure 3.2 as an example, we would get the sum of:

- The difference between branch lengths of A in  $T_1$  (.3) and  $T_2$  (.27), squared;
- The difference between branch lengths of B in  $T_1$  (.2) and  $T_2$  (.15), squared;

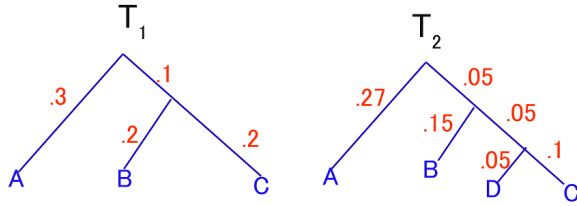


Figure 3.2: BSD Example

- The difference between branch lengths of C in  $T_1$  ( $.1 + .2$ ) and  $T_2$  ( $.05 + .05 + .1$ ), squared; and
- The difference between branch lengths of D in  $T_1$  (0, since the branch does not exist in  $T_1$ ) and  $T_2$  ( $.05$ ), squared.

or:

$$(.3 - .27)^2 + (.2 - .15)^2 + (.3 - .2)^2 + (0 - .05)^2 = .0159 \quad (3.1)$$

The quartets distance (which we did not make use of) is very similar to the RF distance measure. In the RF distance, the taxa (leaf nodes) are separated into two subsets by "cutting" internal edges. In the quartets, the trees are cut into all possible sections of four. The number of quartets that have different topologies, in two unrooted trees of arbitrary degree, is the quartets distance.

For comparing the results of the "pure" maximum likelihood approach and GARLI, we also compared the *log likelihood* score. The log likelihood score (also known as  $\ln(L)$ ) is, as its name implies, the log of the maximum likelihood score; the log is taken to make the ML scores more manageable. To understand why the log is preferred, we need to look into the calculation of the ML score itself.

Given some prediction  $P$  of an event occurring (i.e., rolling the sum of 7 on two six-sided dice), the likelihood score is  $L = \prod_{i=1}^n D(x_i - P)$  where the  $x_i$ 's are observed data (i.e., previous rolls) and  $D(x)$  is an assumed probability distribution. For instance, if we are looking for the likelihood of  $Z$  given a set of data  $X$  and estimated coefficients  $A$  i.e.,

$$Z = A \cdot X + \varepsilon$$

where  $\varepsilon$  is an error term with the assumed probability distribution. This gives us  $D(Z - AX)$  (the probability of  $Z$  given the estimated coefficients and  $X$ ), and therefore  $L = \prod_{i=1}^n D(Z - AX)$  (assuming independence of events). Thus we want to get the estimated coefficients  $A$  as "good" as possible to maximize  $L$ . The  $L$  score becomes smaller and smaller with the addition of more terms/data, i.e. in rolling a sum of 7 on two six-sided dice multiple times, the likelihood for each roll is  $\frac{1}{6}$ . Thus for 5 rolls we have:

$$L = \left(\frac{1}{6}\right)^5 = .0001286$$

and for 7 rolls we get

$$L = \left(\frac{1}{6}\right)^7 = 3.57 \times 10^{-6}$$

Thus, as the "size" of  $X$  increases, the  $L$  score decreases dramatically, hence the need to take a log of the  $L$  score, resulting in the  $\ln(L)$  score. As mentioned earlier it is not appropriate to compare the  $\ln(L)$  scores unless the number of data/observed events is the same for the likelihoods under comparison.

### 3.3 Results

The raw data from the experimental sets are given in Table 3.1 and Table 3.2.

Robinson-Foulds distances and branch score distances were calculated for all trees, using the TREEDIST program in PHYLIP. These are shown in Figure 3.3 through Figure 3.10.

In addition, log likelihood scores were collected from the trees generated from PHYML and GARLI. These scores are shown in Figure 3.10.

### 3.4 Discussion

Users considering the time required to generate the phylogenetic tree to be the primary consideration would be discouraged from using GARLI. Using

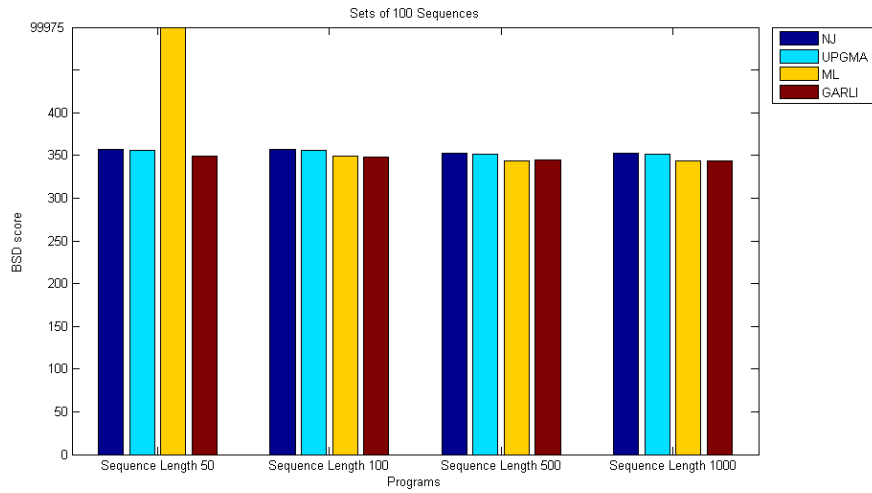


Figure 3.3: Branch Score Distance, 100 sequences

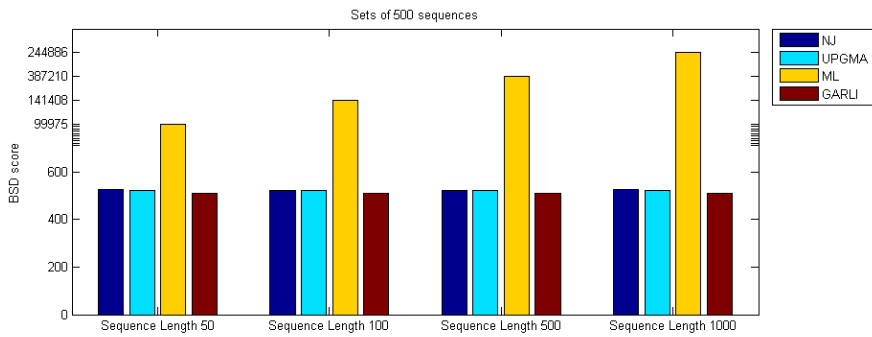


Figure 3.4: Branch Score Distance, 500 sequences

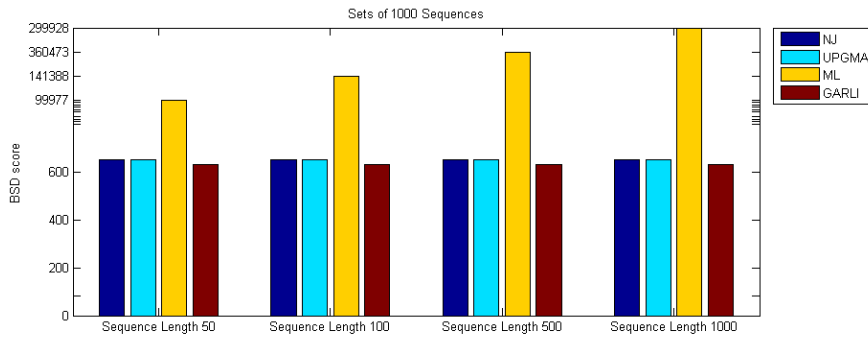


Figure 3.5: Branch Score Distance, 1000 sequences

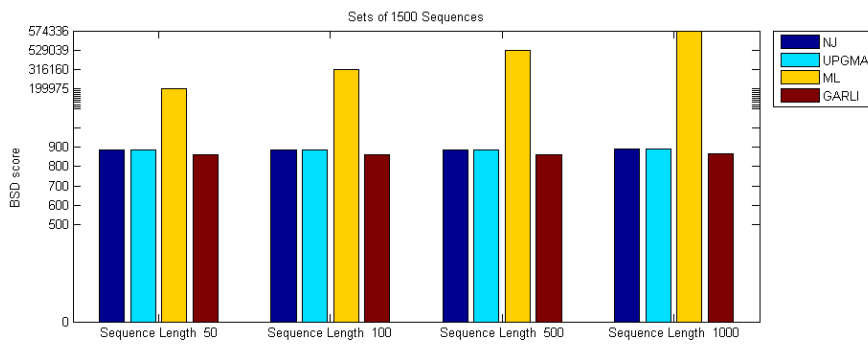


Figure 3.6: Branch Score Distance, 1500 sequences

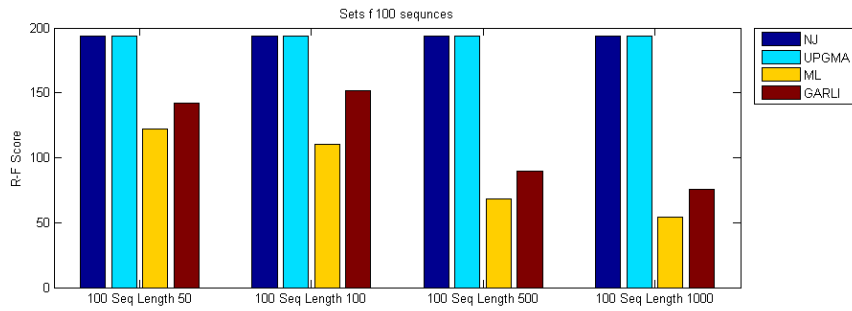


Figure 3.7: Robinson Foulds Distance, 100 sequences

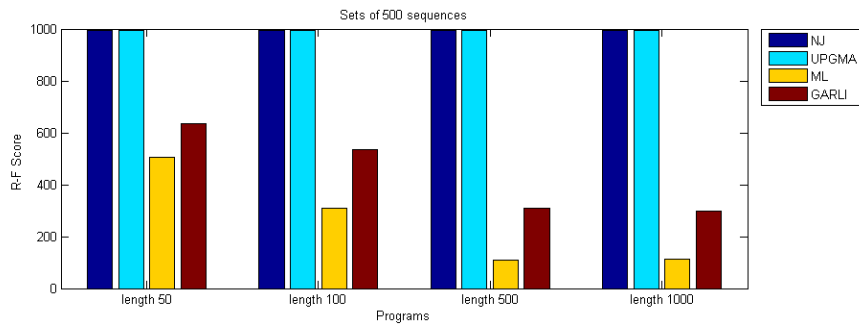


Figure 3.8: Robinson Foulds Distance, 500 sequences

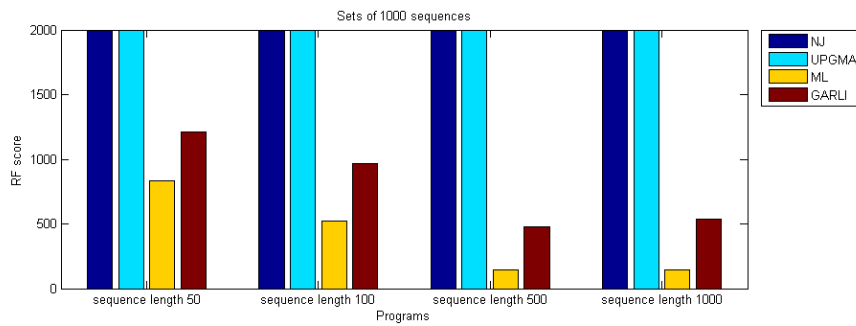


Figure 3.9: Robinson Foulds Distance, 1000 sequences

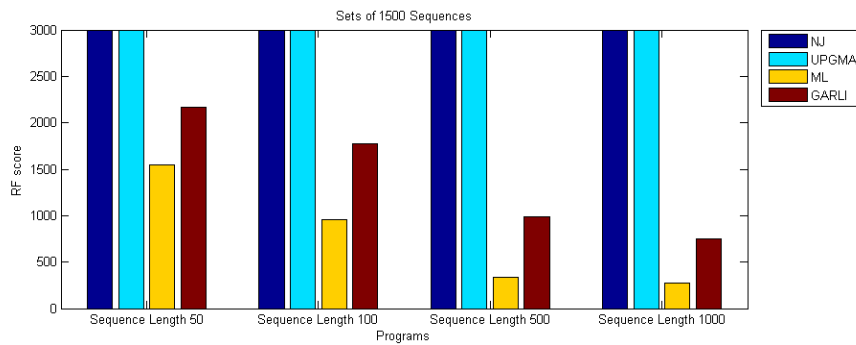


Figure 3.10: Robinson Foulds Distance, 1500 sequences

Set	NJ	UPGMA	PHYML/Hasegawa	GARLI/NJ
NS100L50	0.02	0.02	0.27	12.25
NS100L100	0.02	0.02	0.42	13.73
NS100L500	0.02	0.02	3.82	25.83
NS100L1000	0.02	0.02	10.43	39.83
NS500L50	0.03	0.02	2.98	135.47
NS500L100	0.03	0.02	5.67	148.12
NS500L500	0.03	0.02	55.13	98.33
NS500L1000	0.03	0.02	133.75	215.17
NS1000L50	0.50	0.36	8.40	1150.82
NS1000L100	0.29	0.20	15.43	587.85
NS1000L500	0.32	0.22	67.77	661.53
NS1000L1000	0.30	0.20	211.67	765.98
NS1500L50	1.01	0.69	20.22	2744.58
NS1500L100	1.01	0.68	37.93	2416.40
NS1500L500	1.00	0.67	168.40	2266.03
NS1500L1000	1.02	0.68	262.45	2438.68

Table 3.1: Tree generation times (minutes) for Neighbor Joining (NJ), UPGMA, Maximum Likelihood (ML), GARLI

an NJ tree as the starting topology, as recommended in the User’s Guide, the run time for a data set comparable to the size of the LANL set involved in this project would be approximately 40+ hours.

Users considering only the resulting topology (as expressed by the RF distance) would similarly be discouraged from using GARLI, since GARLI was uniformly unable to achieve the RF scores achieved by PHYML. Even when GARLI was given the PHYML tree as a starting topology, the RF score *increased*.

It is only when comparing on the basis on lnL score that the benefit of GARLI becomes apparent. We believe that this is the correct metric to use in comparing trees, since it takes both topology and branch distance into account (unfortunately, NJ and UPGMA trees do not have a computed lnL score; presumably it can be calculated separately, but we did not do so, disqualifying them as serious competitors based on topology score and literature review.).

It should be noted that when we say that GARLI had better lnL scores,



Set	NJ RF	NJ BSD	UPGMA RF	UPGMA BSD	ML RF	ML BSD	ML lnL	GARLI/NJ RF	GARLI/NJ lnL	GARLI/NJ BSD
NS100L50	194	356.689	194	355.828	122	99990.610	-5555.	142	-5548.167	349.593
NS100L100	194	357.169	194	356.130	110	349.010	-11403.	152	-11387.158	348.628
NS100L500	194	352.820	194	351.728	68	344.163	-58856.	90	-58849.077	344.222
NS100L1000	194	352.417	194	351.176	54	343.486	-116977.	76	-116970.150	343.602
NS500L50	994	523.979	994	523.016	504	99975.300	-25363.	634	-24864.	509.479
NS500L100	994	523.176	994	522.221	310	141408.100	-51951.	534	-50837.	508.196
NS500L500	994	523.380	994	522.200	108	387210.400	-258240.	308	-249025.	509.478
NS500L1000	994	523.984	994	523.013	112	244885.800	-518999.	298	-500897.	510.147
NS1000L50	1994	649.550	1994	648.677	832	99977.990	-47549.	1212	-46451.	631.390
NS1000L100	1994	649.638	1994	648.682	522	141388.800	-95507.	970	-94070.	629.301
NS1000L500	1994	649.716	1994	648.632	146	360473.000	-485123.	478	-468389.	631.148
NS1000L1000	1994	649.794	1994	648.648	148	299928.700	-983791.	534	-949445.	631.212
NS1500L50	2994	885.228	2994	884.287	1548	199975.400	-72712.	2170	-72052.	859.777
NS1500L100	2994	887.058	2994	886.137	952	316160.300	-148640.	1776	-146198.	860.545
NS1500L500	2994	887.391	2994	886.327	332	529039.100	-752226.	990	-728222.	863.222
NS1500L1000	2994	889.347	2994	888.271	270	574336.100	-1502311.	754	-1449398.	865.151

Table 3.2: Robinson Foulds (RF), Branch Score Distance (BSD), Log Likelihood (lnL) Scores

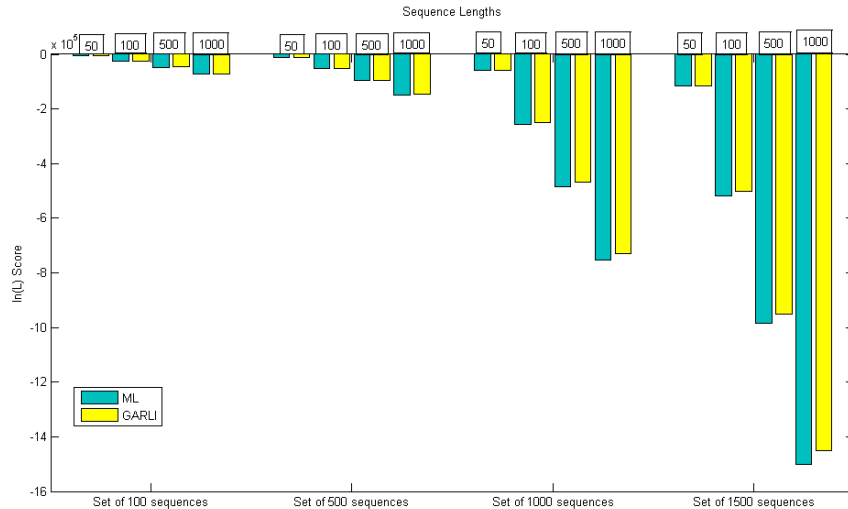


Figure 3.11: Log Likelihood Scores

we are unable to say whether these scores are *significantly* better, a problem noted in Zwickl: “There is not a standard number of lnL units that may be considered significant across topologies because the distribution on lnL scores depends very strongly on the details of the particular dataset.” [59]. Zwickl’s solution was to compare topologies using the Shimodaira-Hasegawa test in PAUP, but he noted that an internal bug prevented it from running on very large datasets. We did not have time to investigate the Shimodaira-Hasegawa test.

### 3.4.1 Algorithms

While neighbor joining and UPGMA are clearly very fast (unsurprising, given their algorithmic approaches), they also produce trees of relatively low accuracy. The poor quality of the results, and the invariability of the results regardless of sequence length are troubling. However, a few spot checks with BIONJ and MatLab created trees that scored identically.

The ML trees produced by PHYML were uniformly superior from a topological perspective, and the quality improved as the number of nucleotides in the sequences increased. Generation times for PHYML increase at a rate

above linear, but below quadratic.

GARLI, using the neighbor-joining tree as the starting point, was unable to achieve RF scores equivalent to the “pure” ML approach. However, it was able to produce the best lnL score. Generation times for GARLI were relatively unpredictable. There appeared to be little correlation between sequence length and generation time with sets, especially as the sets grew larger.

The performance of PHYML was particularly impressive when compared to GARLI, as it was able to produce a more topologically accurate tree in a fraction of the time.

### 3.4.2 Behavior over Sequence and Set Size Increases

For neighbor joining and UPGMA, the quality of the tree did not change with regard to either sequence length or sequence set size. Clearly, using longer sequences did nothing to either increase or decrease tree quality with these two algorithms, since the scores were the same for all sequence lengths within a given set size<sup>1</sup>. When comparing the RF scores of sets of differing size, the straightforward solution is to normalize the RF scores by dividing the score by the associated set size. After normalization, it becomes clear that the change in quality as set size increased was slight ( $194/100 = 1.94$ ;  $994/500 = 1.988$ ;  $1994/1000 = 1.994$ ;  $2994/1500 = 1.996$ ).

For both the “pure” ML approach and the GARLI approach, tree quality improved dramatically as sequence length increased.

For ML and GARLI, the tree quality decline with respect to the sequence set size was approximately linear with very short sequence length, but was impressively sub-linear for longer sequence lengths.

For Neighbor Joining, UPGMA, and GARLI, the BSD score increased linearly as the sequence set size increased. In the case of ML, the BSD score quickly went to a level several orders of magnitude higher than the other programs.

---

<sup>1</sup>The RF scores for Neighbor Joining and UPGMA were seen as suspicious. However, the scores found by TREEDIST were identical to those found by PhyloNet [21], which would appear to eliminate the scoring program as a possible cause. Console output and error files were re-examined, as were the output files generated by NEIGHBOR in the course of creating the Neighbor Joining and UPGMA trees. No errors were found, and the trees appear to have generated correctly. At this point, we are unable to say that the results are incorrect...but we’re still suspicious.

## 3.5 Future Work

As is reportedly typical for Math Clinics, ideas are encountered late in the semester which suggest interesting directions for further work, or different paths that may have optimized the project work undertaken. This team’s experience has been no different: in this section, we present some additional experiments that point to paths not taken.

### 3.5.1 Using Better Start Trees in GARLI

The GARLI user manual [58] suggests the use of a starting topology, and specifically states that “[i]t does not appear to make much of a difference how these topologies are obtained.” The length of time required to generate trees topologically inferior to those generated using a “pure” ML approach led us to a question: would providing the ML tree generated by PHYML as the starting topology cause GARLI to arrive at a better result?

From the four cases evaluated, the answer appears to be positive. Using the ML topology, GARLI arrived at a “better” answer in much less time. The significance of this is the identification of a means of lowering the GARLI run time for a similar or better result; the time required for a combined PHYML run and GARLI run using the resulting tree is significantly less than that required to run GARLI with the NJ starting topology.

On the other hand, Zwickl [59] has stated concerns that starting with a near-optimal starting topology may prematurely lock GARLI into a local optimum. Whether this is the case here is unclear, though, if so, the optimum found is (at least minutely) better than the optimum found via the use of NJ as the starting topology. It should also be noted (again, from four data points) that using the ML tree that assumed the Jukes-Cantor, rather than the Hasegawa, model as the starting point for GARLI, and using a **ratematrix** setting of *irate* (an assumption that substitutions between pairs always occurs at the same rate, corresponding to the Jukes-Cantor evolutionary model) resulted in significant improvements in the RF (topology) score, while holding the lnL score relatively steady (see Table 3.4).

In summary, the question of which tree to use as a starting point for GARLI (or using a random start, rather than a tree) seems worth of investigation: what tree would put GARLI on the best path toward a solution, thus minimizing run time, without prejudicing it toward a local optima?

### 3.5.2 Selection of Evolutionary Model

The second issue involves the consideration of the evolutionary model when generating trees. As mentioned above, the experimental data sets were created using a Jukes-Cantor evolutionary model. This fact was immaterial in the generation of NJ and UPGMA trees, but potentially significant in the generation of ML and GARLI trees.

As the above results show, moving from the default Hasegawa model to the Jukes-Cantor model in PHYML has significant apparent impacts in relation to larger sequence sets, *if* the underlying evolutionary model is, in fact, Jukes-Cantor. This statement is a result of only four data points, but suggests an interesting direction for further investigation: when the evolutionary model is unknown, which should be chosen to maximize accuracy?

### 3.5.3 Improved Experimental Design

The non-deterministic nature of the the generation of data sets using ROSE was potentially problematic: the simulated data set may have contained elements that, while possible, served to lead the multiple sequence alignment and/or the tree generation astray.

A better design may have been to use the original tree as input to ROSE in the creation of multiple data sets. The experiments could then be run across all the members of the set, and the distribution of scores analyzed. This would minimize the potential of a single, atypical data set to skew the results.

## 3.6 Conclusions

Based on our experimental findings, we conclude that, for the LANL data set that is the subject of the broader Math Clinic study, GARLI would be most likely to generate the most accurate tree. However, it is computationally very expensive. PHYML is able to produce a tree of comparable accuracy in a significantly shorter period of time.

As per the GARLI user manual, a starting topology should be provided. We have used the NJ tree, generated via the DNADIST and NEIGHBOR programs in PHYLIP, but MatLab may also be used.

We have had promising indications that using the ML tree as the starting topology results in a shorter GARLI running time, without compromising the

quality of the result; however, this would obviously involve the creation of a maximum likelihood tree, itself a time-consuming proposition for a data set of this size.

Set	NJ Time	RF/ NJ	lnL/ NJ	BSD/ NJ	ML Time	RF/ ML	lnL/ ML	BSD/ ML
NS100L1000	39.83	76	-116970.	343.	15.97	68	-116970.	343.
NS500L1000	215.17	298	-500897.	510.	99.55	242	-501810.	508.
NS1000L1000	765.98	534	-949445.	631.	324.05	382	-947735.	629.
NS1500L1000	2438.68	754	-1449398.	865.	1260.98	534	-1446348.	862.

Table 3.3: GARLI generation times (minutes) and scoring, using Neighbor Joining (NJ) and Maximum Likelihood (ML) starting topologies

Set	ML/ H/RF	ML/ H/lnL	ML/ H/BSD	ML/ JC/RF	ML/ JC/lnL	ML/ JC/BSD
NS100L1000	68	-116970.	343.421	64	-116992.	343.558
NS500L1000	242	-501810.	508.493	164	-501837.	508.494
NS1000L1000	382	-947735.	629.290	202	-947728.	629.249
NS1500L1000	534	-1446348.	862.301	254	-1446063.	862.562

Table 3.4: GARLI scoring, using Maximum Likelihood (ML) starting topologies from Hasegawa (H) and Jukes-Cantor (JC)

Set	ML/ RF/H	ML/ BSD/H	ML/ lnL/H	ML/ RF/JC	ML/ BSD/JC	ML/ lnL/JC
NS100L1000	54	343.486	-116977.	54	343.502	-117001.
NS500L1000	112	244885.800	-518999.	80	509.360	-500345.
NS1000L1000	148	299928.700	-983791.	94	99977.990	-946175.
NS1500L1000	270	574336.100	-1502311.	154	864.314	-1444535.

Table 3.5: Maximum Likelihood (ML) scoring, using Hasegawa (H) and Jukes-Cantor (JC) Evolutionary Models

### 3.A Generating Sequence Data

```
#!/usr/bin/perl -w
#####
# genDataSet.pl
#
# Jeff Kenyon
# Math Clinic, Fall 2007
#
# Generate a set of sequences of a given size, with
# an approximate specified length. Output a text
# file of sequences in FASTA format.
#####

use strict;
use Getopt::Long;

use Bio::AlignIO;
use Bio::Seq;
use Bio::SeqIO;

use Math::Random::OO::Normal;
use POSIX;

# Argument defaults
my $setSize = 100;
my $seqLen = 100;
# For mutationFreq, it's a percentage:
# i.e., .05 = for every 100 nucleotides, 5 mutations
# Per Yana: 10E-4
my $mutationFreq = .0001;
# Default sequence is a portion of the sequenced E. coli:
# http://www.genome.wisc.edu/pub/sequence/AE005174v2.fas
my $rootSeq = "ecoli.fas";

GetOptions("setSize=i" => \$setSize, # --setSize
          "seqLen=i" => \$seqLen, # --seqLen
          "rootSeq:s" => \$rootSeq, # --rootSeq
```

```

        "mutationRate:f" => \$mutationFreq
    );

    # Generation parameters
    my $offspringPerGen = 5;
    # originally thought of having random insertion/deletion size,
    # but then thought doing it on an AA basis might be better.
    my $insertionSize = 3;
    my $deletionSize = 1;
    my $generation = 0;
    my $sequenceCount = 0;
    my $sequence;
    my $avgMutationsPerSequence = ceil($seqLen*$mutationFreq);
    my $mutationStdDev = ceil($avgMutationsPerSequence * .1);
    my $dist =
        Math::Random::OO::Normal->new($avgMutationsPerSequence,
                                       $mutationStdDev);
    my @nucleotideChoices = ("A","C","T","G");

    print "Avg. mutations per sequence = " .
        "$avgMutationsPerSequence, StdDev = $mutationStdDev\n";

    my @sequences;
    my @genStart;
    my @genEnd;

    # open up root sequence file, read in $seqLen characters.
    open(SEQ,"< $rootSeq") or
    die "Root sequence file $rootSeq not found: $!\n";
    # read the header line to get it out of the way
    while(<SEQ>) {
        my $line = $_;
        if ($line =~ /^>/) {
            # header line...skip it
            next;
        }
        # trim the line feed
        chop($line);
    }

```



```

    $sequence = $sequence . uc($line);
    if (length($sequence) > $seqLen) {
        last;
    }
}
close(SEQ);
# trim to the exact length desired
$sequence = substr($sequence,0,$seqLen);

# add the root sequence to the sequences generated
$sequences[$sequenceCount] =
    Bio::Seq->new(-display_id => 'MC000000',
        -accession_number => 'MC000000',
        -desc => "A/simulated/US/-1/0/2007 length: $seqLen",
        -seq => $sequence);
$genStart[0] = 0;
$genEnd[0] = 1;

$sequenceCount++;

#print $sequences[0]->seq,"\n";

# loop till we've generated enough sequences
while ($sequenceCount < $setSize) {
    $generation++;
    print "Generation $generation\n";
    $genStart[$generation] = $sequenceCount;
    for (my $i = $genStart[$generation-1];
        $i < $genEnd[$generation-1]; $i++) {
        offspring($sequences[$i],$i);
    }
    $genEnd[$generation] = $sequenceCount;
    print "Sequence count at end of generation " .
        "$generation = $sequenceCount\n";
}

# Go through last generation and trim some, for exact number
my $sequencesOverLimit = $sequenceCount - $setSize;

```

```

my $numberInGeneration =
    $genEnd[$generation] - $genStart[$generation];
while ($sequenceCount > $setSize) {
    for (my $i = $genStart[$generation];
         $i < $genEnd[$generation]; $i++) {
        if (rand(1) < $sequencesOverLimit/$numberInGeneration) {
            if ($sequences[$i] != -1) {
# print "Deleting $i\n";
                $sequences[$i] = -1;
                $sequenceCount--;
            }
        }
        if ($sequenceCount == $setSize) {
            last;
        }
    }
}

print "Sequence count at end of trimming = $sequenceCount\n";

# Output FASTA sequences
my $out = Bio::SeqIO->new(-file => ">fasta.out.txt" ,
                        '-format' => 'Fasta');
for (my $i = 0; $i <= $#sequences; $i++) {

    if ($sequences[$i] != -1) {
        $out->write_seq($sequences[$i]);
    }
}

#####
# SUBROUTINES
#####

sub offspring {
    my $nucleotideSeq = $_[0]->seq;
    my $parentSeq = $_[1];
    my $mutations = ceil($dist->next());

```

```

my $numberOfOffspring = int(rand $offspringPerGen) + 1;
for (my $x = 0; $x < $numberOfOffspring; $x++) {
    my $newSeqString = $nucleotideSeq;
    for (my $i = 1; $i <= $mutations; $i++) {

        # roll the die: what kind of mutation will it be?
        my $roll = int(rand 3) + 1;
        if ($roll == 1) {
# swap
# position of nucleotide to swap
my $pos = int(rand length($newSeqString));
my $charAtPos = substr($newSeqString,$pos,1);
my $newCharAtPos = $charAtPos;
# find a different nucleotide
while ($charAtPos eq $newCharAtPos) {
    $roll = int(rand 4);
    $newCharAtPos = $nucleotideChoices[$roll];
}
substr($newSeqString,$pos,1) = $newCharAtPos;

        } elsif ($roll == 2) {
# insertion: make up a sequence of three nucleotides,
    # then insert it
my $insertion = "";
for (my $j = 0; $j < 3; $j++) {
    $roll = int(rand 4);
    $insertion = $insertion . $nucleotideChoices[$roll];
}
my $pos = int(rand length($newSeqString));
$newSeqString =
    substr($newSeqString,0,$pos) . $insertion .
        substr($newSeqString,$pos);

        } elsif ($roll == 3) {
# deletion
my $pos = int(rand length($newSeqString)-2);
while ($pos % 3 != 0) {

```

```

    $pos++;
}
if ($pos+3 < length($newSeqString)) {
    $newSeqString =
        substr($newSeqString,0,$pos).
            substr($newSeqString,$pos+3);
} else {
    # $pos is the last character
    $newSeqString = substr($newSeqString,0,$pos);
}

    } else {
die "Our die is broken!! The roll was $roll!!\n";
    }

}
my $newSeq = Bio::Seq->new();
$newSeq->seq($newSeqString);
$newSeq->display_id(sprintf("MC%02d%04d",
                            $generation,$sequenceCount));
$newSeq->accession_number(sprintf("MC%02d%04d",
                                $generation,$sequenceCount));
$newSeq->desc("A/simulated/US/$parentSeq/$mutations/2007" .
            " length:" . length($newSeqString));
# print $newSeq->display_id()," : ", $newSeq->seq, "\n";
$sequences[$sequenceCount++] = $newSeq;
}
}

# FASTA FORMAT
# -- Header line does not have a canonical form, but
# we'll be using a form matching the LANL data:
# >accession strain serotype length:xxxx
# -- Accession is typically two letters/6 digits; we'll
# be using MC (Math Clinic) as our letters, first
# two are generation number, last four are unique
# index number
# -- strain will be

```

```
# A/simulated/US/parent_id/number_of_mutation/2007
# -- serotype will always be H5N1
# -- Sequence lines are 76 characters long

0;
```

### 3.B Using ROSE to Generate Sequences and Trees

ROSE is describe in [46], and elsewhere in this report, so it will not be discussed in detail here.

ROSE has a large number of parameters for generating sequence data, few of which were used in generating the sequence data sets for this experiment.

Since no tree was specified, ROSE created a uniform binary tree. There were no insertions and deletions (indels) in the data set, which in retrospect, may not have been the best setting.

The Jukes-Cantor evolutionary model was used.

The following code is the content of the input file for ROSE. Only the *SequenceLen* and *SequenceNum* arguments were altered in generating the 16 data sets.

The program itself may be downloaded from  
<http://bibiserv.techfak.uni-bielefeld.de/rose/>.

```
# rose adapted from rose example/sample 4

#include dna-defaults

SequenceLen = 1000
SequenceNum = 1500
Relatedness = 250
ChooseFromLeaves = True
TreeWithAncestors = True

TheInsertThreshold = 0.00005
TheDeleteThreshold = 0.00005

TheInsFunc = [.2,.3,.4,.4,.3,.2,.1]
```

```
TheDelFunc = [.2,.3,.4,.4,.3,.2,.1]
```

```
# my additions
```

```
OutputFilebase = "output"
```

```
TheDNAModel = "JC"
```

```
StdOut = False
```

# Chapter 4

## The Effect of Tree Topology on Leaf Distances

By Angela Harris, Craig Tennenhouse and Michael Trujillo

### Abstract

In this paper we analyze the effect that different tree topologies have on pairwise distances between nodes of the trees.

### 4.1 Introduction

A phylogenetic tree is a diagram that shows evolutionary relationships of organisms. The nodes represent the taxonomic units, and the branches define the relationships among the units in terms of descent and ancestry. The branching pattern of the tree is called the topology. Between any pair of taxonomic units we can determine a distance that relates how "different" the units are. We can use these distances to assign lengths to each branch of the tree. Two trees are considered to be topologically equivalent if they have the same branching pattern, regardless of the lengths of the branches.

In this project the nodes of the tree represent the mutations of the H5N1 virus, also known as the "bird flu". One of the goals of the project is to determine a rate of change of these mutations. In order to do this, we must produce a phylogenetic tree for the data set. There are several methods for producing phylogenetic trees, each potentially producing a different tree topology. In

order for the rate of change of the virus to be determined accurately, it is necessary to obtain accurate distances between each pair of nodes on the tree. Our objective is to determine whether the tree topology has an effect on these pairwise distances. In other words, do different tree topologies on the same data give different pairwise distances between taxa?

The answer to the above question will help determine which tree building method to use. Two of the considerations that contribute to the decision of which tree building method to use are the speed of the algorithm and the accuracy of the algorithm. The methods that produce the more accurate tree topologies use the faster tree building algorithms to generate a tree quickly, and then spend a good deal of time trying to refine that tree, searching for a topology that better fits the data. If different topologies representing the same set of data yield approximately the same pairwise distances between nodes, then building the correct tree topology is not necessary for obtaining an accurate rate of change for the nucleotide sequences. This would mean that a faster algorithm could be used for tree construction, while still obtaining an accurate rate of change.

## 4.2 The Procedure

Our task is to determine whether different tree topologies give different pairwise branch distances on the same data. Since the outcome of this study would potentially effect the tree building method used for the H5N1 data, we used industry accepted algorithms to build our trees rather than just producing random trees. Trees built by the programs in PHYLIP [12] are likely evolutionary trees, whereas random trees may not be. We also decided not to create a tree and make random changes to its topology for a similar reason. Many of the random changes that could be made to a tree would not make sense in an evolutionary tree. Hence, we used tree building programs to generate different topologies for the same data set.

We give an overview of our procedures here, which followed the chart in Figure 4.1. For details of how the project was carried out see 4.A. We created a tree to be the "true" tree, referred to as the original tree throughout the paper. This gave us a standard by which to compare the tree building methods. This tree was built in MATLAB, where the pairwise distances between each pair of nodes was calculated. These trees were input into a program called Dawg [2], which assigned DNA sequences to each tree. These



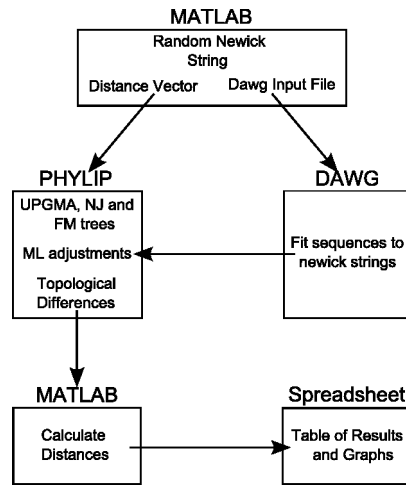


Figure 4.1: Analysis Flowchart

sequences along with the pairwise distances from MATLAB were input into the programs Fitch, Neighbor, and Dnaml in PHYLIP.

Fitch and Neighbor were used to produce three different trees, using the distances produced in MATLAB. The three trees were produced using the Fitch-Margoliash, Neighbor-Joining and UPGMA algorithms. UPGMA is the simplest of all tree building methods, and is the fastest. Neighbor-Joining is also a fast algorithm, but is able to obtain a more accurate tree topology much more often than UPGMA. The Fitch-Margoliash algorithm is the slowest of the three, but is also the most accurate. (For comparison of these algorithms (and others) see [44].) Dnaml was used to adjust branch lengths on each of these trees, using the DNA sequences provided by Dawg. Notice that we did not use Dnaml to produce a tree or to alter a tree topology. The trees that were produced by Dnaml had the same topology with which they entered Dnaml, only the branch lengths were changed.

We then compared the topologies of the original tree and the three trees built from its distance data by Fitch and Neighbor. We chose to use the symmetric difference comparison in Treedist, another program in PHYLIP, to determine the differences in the tree topologies. This comparison was done pairwise, so that the three trees created in PHYLIP were compared to each other, as well as to the original tree.

Finally, we sent all the data obtained in PHYLIP back into MATLAB

where the distances between the original tree and all trees created from its data were measured. We decided to use three different methods of measure: maximum distance, mean distance, and precedence distance. For a definition of these measures, see Section 4.A. We compared the distances with the topology differences to determine the effect that tree topology had on pairwise branch distances.

### 4.3 Results

For each of our original trees, the Fitch-Margoliash and Neighbor-Joining algorithms built a tree with the same topology. They also reproduced the pairwise branch distances to within  $1.2 \times 10^{-4}$ , with the mean difference being less than  $4 \times 10^{-5}$  (see Figures 4.2, 4.3). When we consider the precedence distance, we see that there is almost no difference between the original and the Fitch-Margoliash tree in most cases, as shown in Figure 4.4. For the Neighbor-Joining tree, this distance is very small, but does not approach zero, which is shown in Figure 4.5. The numerical results for the Fitch-Margoliash and Neighbor-Joining trees are shown in Figures 4.6 and 4.7, respectively.

The UPGMA trees were quite different. The UPGMA algorithm built trees that differed in topology from the original. The Robinson-Foulds distance between the original trees and those built by the UPGMA algorithm are shown in Figure 4.8. We also found that the Robinson-Foulds distance increased linearly with the number of leaves, as can be seen in Figure 4.9. All three measures of the pairwise branch differences were significantly higher than those of the Fitch-Margoliash and Neighbor-Joining trees. The maximum and mean distances, shown in Figure 4.10, increase as the number of nodes of the tree increases, though it is difficult to tell with such a limited data set if the increase is linear, sub-linear, or other. Interestingly enough, the precedence distance for the UPGMA trees, though much higher than the other methods, stays in the range of 0.2 – 0.3, for the most part (see Figure 4.11). Numerical results for the UPGMA trees is shown in Figure 4.12.

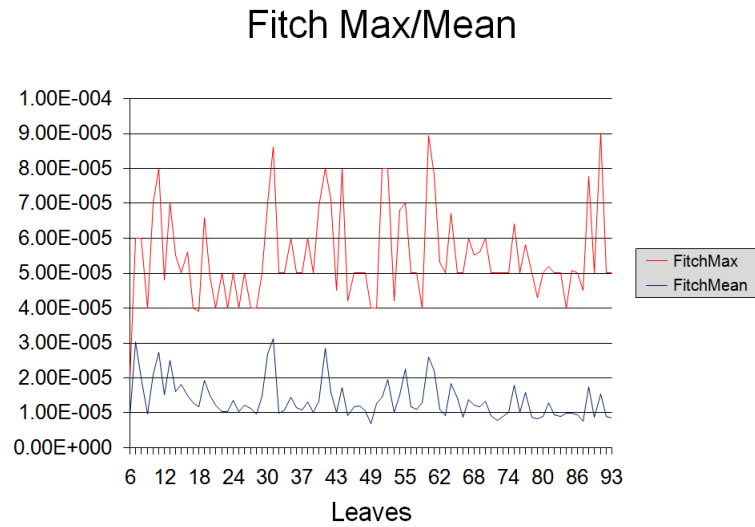


Figure 4.2: Accuracy of Pairwise Branch Distances (Fitch-Margoliash)

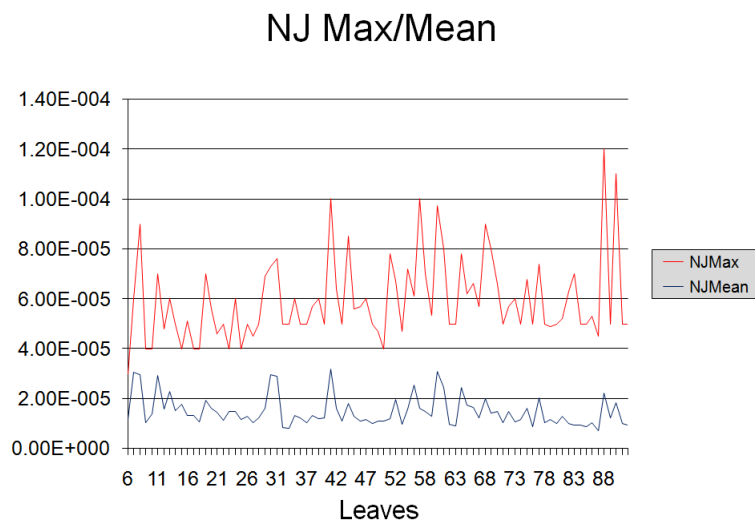


Figure 4.3: Accuracy of Pairwise Branch Distances (Neighbor Joining)

### Fitch Precedence

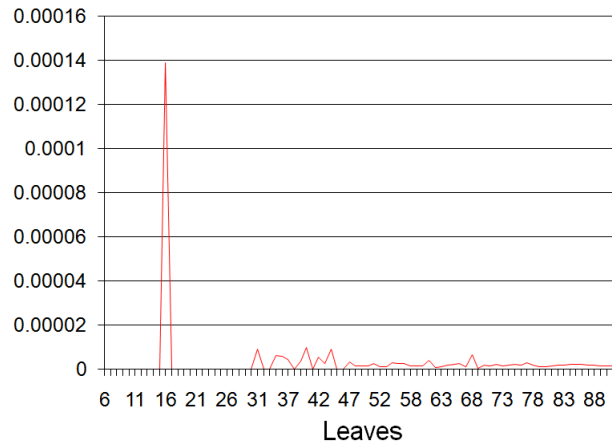


Figure 4.4: Precedence Metric, (Fitch-Margoliash)

### NJ Precedence

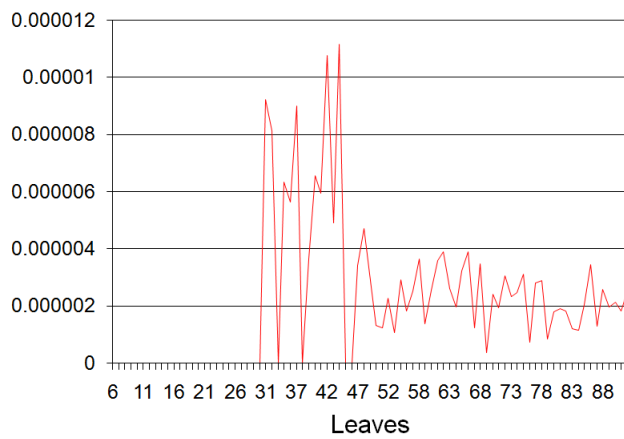


Figure 4.5: Precedence Metric, (Neighbor Joining)

Leaves	FitchMax	FitchMean	FitchPrec	Leaves	FitchMax	FitchMean	FitchPrec
6	2.00E-005	9.33E-006	0	56	5.00E-005	1.18E-005	2.53E-006
7	6.00E-005	3.05E-005	0	58	5.00E-005	1.12E-005	1.46E-006
8	6.00E-005	2.00E-005	0	59	4.00E-005	1.29E-005	1.37E-006
9	4.00E-005	9.72E-006	0	60	8.92E-005	2.60E-005	1.28E-006
10	7.00E-005	2.09E-005	0	61	7.80E-005	2.21E-005	4.18E-006
11	8.00E-005	2.75E-005	0	62	5.30E-005	1.10E-005	5.59E-007
12	4.80E-005	1.53E-005	0	63	5.00E-005	9.37E-006	1.05E-006
13	7.00E-005	2.52E-005	0	64	6.70E-005	1.84E-005	1.97E-006
14	5.50E-005	1.62E-005	0	65	5.00E-005	1.43E-005	2.31E-006
15	5.00E-005	1.83E-005	0	66	5.00E-005	8.74E-006	2.61E-006
16	5.60E-005	1.49E-005	0.00013889	67	6.00E-005	1.38E-005	1.23E-006
17	4.00E-005	1.30E-005	0	68	5.50E-005	1.23E-005	6.55E-006
18	3.90E-005	1.19E-005	0	69	5.60E-005	1.17E-005	3.63E-007
19	6.60E-005	1.94E-005	0	70	6.00E-005	1.34E-005	1.71E-006
20	4.90E-005	1.51E-005	0	71	5.00E-005	9.34E-006	1.30E-006
21	4.00E-005	1.22E-005	0	72	5.00E-005	7.82E-006	2.14E-006
22	5.00E-005	1.03E-005	0	73	5.00E-005	9.00E-006	1.45E-006
23	4.00E-005	1.04E-005	0	74	5.00E-005	1.02E-005	1.64E-006
24	5.00E-005	1.36E-005	0	75	6.40E-005	1.81E-005	2.34E-006
25	4.00E-005	1.03E-005	0	76	5.00E-005	1.02E-005	1.72E-006
26	5.00E-005	1.23E-005	0	77	5.80E-005	1.58E-005	2.80E-006
27	4.00E-005	1.13E-005	0	78	5.00E-005	8.81E-006	1.77E-006
28	4.00E-005	9.73E-006	0	79	4.30E-005	8.24E-006	1.05E-006
29	5.00E-005	1.47E-005	0	80	5.00E-005	8.94E-006	1.20E-006
30	7.00E-005	2.69E-005	0	81	5.20E-005	1.29E-005	1.52E-006
31	8.60E-005	3.13E-005	9.25E-006	82	5.00E-005	9.60E-006	1.81E-006
32	5.00E-005	9.95E-006	0	83	5.00E-005	9.01E-006	1.73E-006
33	5.00E-005	1.10E-005	0	84	4.00E-005	9.95E-006	2.30E-006
34	6.00E-005	1.46E-005	6.35E-006	85	5.07E-005	9.84E-006	2.04E-006
35	5.00E-005	1.16E-005	5.65E-006	86	5.00E-005	9.46E-006	2.25E-006
37	5.00E-005	1.08E-005	4.51E-006	87	4.50E-005	7.68E-006	1.71E-006
38	6.00E-005	1.31E-005	0	88	7.75E-005	1.76E-005	1.91E-006
39	5.00E-005	1.02E-005	3.64E-006	89	5.00E-005	8.71E-006	1.30E-006
40	6.90E-005	1.34E-005	9.86E-006	90	9.00E-005	1.56E-005	1.62E-006
41	8.00E-005	2.85E-005	0	92	5.00E-005	9.07E-006	1.37E-006
42	7.10E-005	1.59E-005	5.40E-006	93	5.00E-005	8.55E-006	2.08E-006
43	4.50E-005	1.02E-005	2.45E-006	94	6.00E-005	1.20E-005	2.09E-006
44	8.00E-005	1.73E-005	8.94E-006	95	5.00E-005	9.97E-006	1.00E-006
45	4.20E-005	9.32E-006	0	96	5.00E-005	1.09E-005	2.12E-006
46	5.00E-005	1.17E-005	0	97	5.00E-005	1.14E-005	2.03E-006
47	5.00E-005	1.21E-005	3.42E-006	98	5.10E-005	1.32E-005	2.30E-006
48	5.00E-005	1.06E-005	1.57E-006	99	6.62E-005	1.35E-005	1.10E-006
49	4.00E-005	6.97E-006	1.45E-006	100	5.00E-005	8.76E-006	2.12E-006
50	4.00E-005	1.28E-005	1.33E-006	101	0.0001	3.00E-005	2.67E-006
51	8.00E-005	1.44E-005	2.46E-006	102	7.50E-005	2.41E-005	2.79E-006
52	8.00E-005	1.95E-005	1.14E-006	103	6.30E-005	1.09E-005	2.97E-006
53	4.20E-005	1.03E-005	1.05E-006	104	0.0001232	3.91E-005	3.76E-006
54	6.80E-005	1.53E-005	2.93E-006	105	7.10E-005	2.42E-005	3.35E-006
55	7.00E-005	2.26E-005	2.72E-006				

Figure 4.6: Distance Metrics (Fitch-Margoliash)

Leaves	NJMax	NJMean	NJPrec	Leaves	NJMax	NJMean	NJPrec
6	3.00E-005	1.20E-005	0	56	0.0001	1.61E-005	2.53E-006
7	6.00E-005	3.05E-005	0	58	7.00E-005	1.49E-005	3.66E-006
8	9.00E-005	2.96E-005	0	59	5.33E-005	1.28E-005	1.37E-006
9	4.00E-005	1.06E-005	0	60	9.72E-005	3.07E-005	2.55E-006
10	4.00E-005	1.38E-005	0	61	8.00E-005	2.44E-005	3.58E-006
11	7.00E-005	2.93E-005	0	62	5.00E-005	9.69E-006	3.92E-006
12	4.80E-005	1.60E-005	0	63	5.00E-005	9.17E-006	2.62E-006
13	6.00E-005	2.28E-005	0	64	7.80E-005	2.46E-005	1.97E-006
14	5.00E-005	1.53E-005	0	65	6.20E-005	1.74E-005	3.24E-006
15	4.00E-005	1.78E-005	0	66	6.62E-005	1.65E-005	3.91E-006
16	5.10E-005	1.34E-005	0	67	5.70E-005	1.25E-005	1.23E-006
17	4.00E-005	1.33E-005	0	68	9.00E-005	1.99E-005	3.47E-006
18	4.00E-005	1.08E-005	0	69	8.00E-005	1.41E-005	3.63E-007
19	7.00E-005	1.93E-005	0	70	6.60E-005	1.50E-005	2.40E-006
20	5.60E-005	1.63E-005	0	71	5.00E-005	1.05E-005	1.94E-006
21	4.60E-005	1.47E-005	0	72	5.70E-005	1.48E-005	3.06E-006
22	5.00E-005	1.13E-005	0	73	6.01E-005	1.09E-005	2.32E-006
23	4.00E-005	1.50E-005	0	74	5.00E-005	1.16E-005	2.47E-006
24	6.00E-005	1.48E-005	0	75	6.78E-005	1.62E-005	3.12E-006
25	4.00E-005	1.17E-005	0	76	5.00E-005	8.93E-006	7.39E-007
26	5.00E-005	1.29E-005	0	77	7.40E-005	2.02E-005	2.80E-006
27	4.50E-005	1.05E-005	0	78	5.00E-005	1.03E-005	2.88E-006
28	5.00E-005	1.24E-005	0	79	4.90E-005	1.18E-005	8.43E-007
29	6.90E-005	1.61E-005	0	80	5.00E-005	1.02E-005	1.80E-006
30	7.30E-005	2.94E-005	0	81	5.20E-005	1.30E-005	1.91E-006
31	7.60E-005	2.88E-005	9.25E-006	82	6.30E-005	9.98E-006	1.81E-006
32	5.00E-005	8.61E-006	8.13E-006	83	7.00E-005	9.47E-006	1.21E-006
33	5.00E-005	8.06E-006	0	84	5.00E-005	9.42E-006	1.15E-006
34	6.00E-005	1.34E-005	6.35E-006	85	5.00E-005	8.73E-006	2.04E-006
35	5.00E-005	1.23E-005	5.65E-006	86	5.30E-005	1.03E-005	3.44E-006
37	5.00E-005	1.06E-005	9.02E-006	87	4.50E-005	7.24E-006	1.29E-006
38	5.70E-005	1.33E-005	0	88	0.00012	2.24E-005	2.59E-006
39	6.00E-005	1.20E-005	3.64E-006	89	5.00E-005	1.23E-005	1.96E-006
40	5.00E-005	1.22E-005	6.57E-006	90	0.00011	1.83E-005	2.12E-006
41	0.0001	3.18E-005	5.95E-006	92	5.00E-005	1.02E-005	1.83E-006
42	6.41E-005	1.61E-005	1.08E-005	93	5.00E-005	9.43E-006	2.51E-006
43	5.00E-005	1.10E-005	4.91E-006	94	8.12E-005	1.57E-005	2.93E-006
44	8.50E-005	1.82E-005	1.12E-005	95	5.20E-005	1.28E-005	1.40E-006
45	5.60E-005	1.29E-005	0	96	7.00E-005	1.25E-005	1.92E-006
46	5.70E-005	1.10E-005	0	97	5.00E-005	1.15E-005	2.40E-006
47	6.00E-005	1.17E-005	3.42E-006	98	6.00E-005	1.22E-005	2.30E-006
48	5.00E-005	1.03E-005	4.72E-006	99	5.70E-005	1.27E-005	1.95E-006
49	4.70E-005	1.11E-005	2.89E-006	100	5.00E-005	9.41E-006	2.04E-006
50	4.00E-005	1.11E-005	1.33E-006	101	9.00E-005	2.35E-005	3.14E-006
51	7.80E-005	1.21E-005	1.23E-006	102	7.69E-005	2.69E-005	3.24E-006
52	6.70E-005	1.96E-005	2.28E-006	103	6.00E-005	9.70E-006	2.46E-006
53	4.70E-005	9.72E-006	1.05E-006	104	9.32E-005	2.67E-005	4.25E-006
54	7.20E-005	1.58E-005	2.93E-006	105	7.10E-005	2.31E-005	3.49E-006
55	6.10E-005	2.54E-005	1.81E-006				

Figure 4.7: Distance Metrics (Neighbor Joining)

### UPGMA Precedence Distance vs. Topo Distance

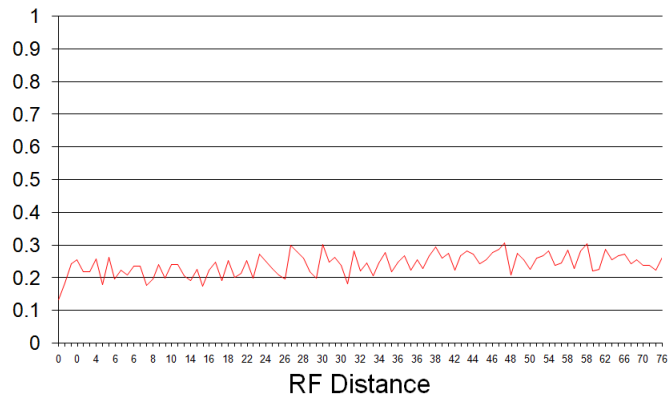


Figure 4.8: Precedence Distance vs. Topo Distance

### RF Dist. vs. Number of Leaves

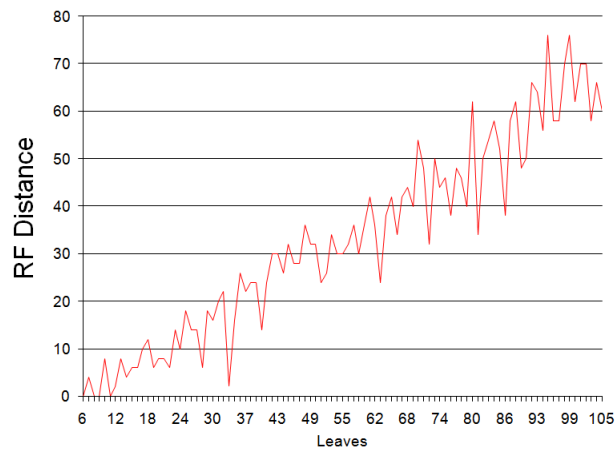


Figure 4.9: RF Dist. vs. Number of Leaves

## UPGMA Max/Mean

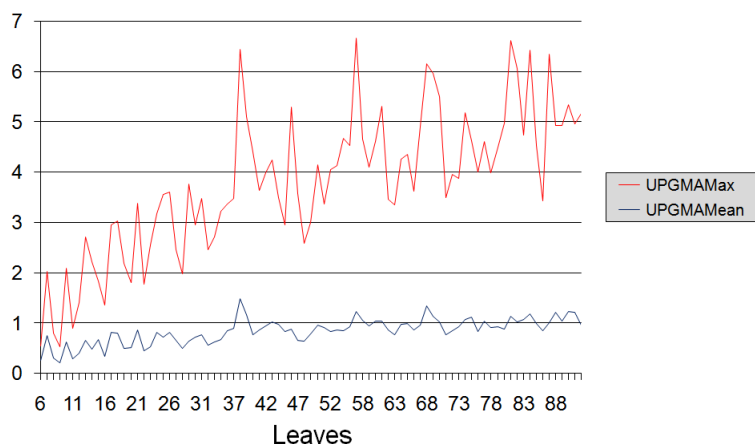


Figure 4.10: UPGMA Max/Mean

## 4.4 Conclusion

Based on our data, we believe that finding the correct tree topology is important in determining the rate of evolution of a species. The only trees that gave us a different topology from our original tree were those created by UPGMA. As the number of nodes on the trees increased, the difference in the pairwise distances between the nodes of the UPGMA trees increased. With the trees produced by the Fitch-Margoliash and Neighbor-Joining algorithms, the difference in the pairwise distances was fairly constant.

When we used the Dnaml program on the sequences provided by Dawg, with the trees built in PHYLIP as the user trees, the product was trees with pairwise branch distances farther from the original tree than before Dnaml was run. We were greatly confused by this at first, since the purpose of performing a maximum likelihood algorithm on a tree is to improve the branch lengths. We now believe that this occurred because we had exact distances for constructing the original trees, while the sequences generated by Dawg were not directly correlated with these distances, but were randomly generated to fit them. This is a hypothesis that we would like to test in the future by creating a tree, fitting sequences to that tree, finding the distances



## UPGMA Precedence

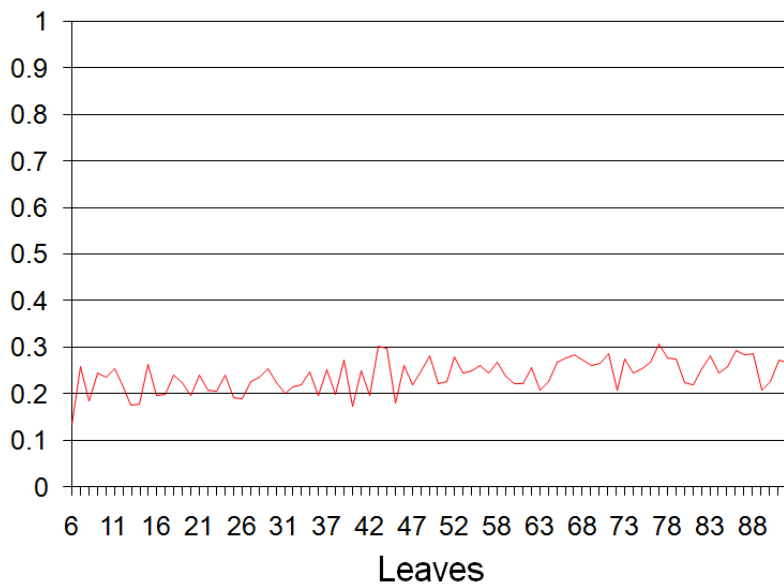


Figure 4.11: UPGMA Precedence

Leaves	UPGMAMax	UPGMAMean	UPGMAPrec	UPGMARF	Leaves	UPGMAMax	UPGMAMean	UPGMAPrec	UPGMARF
6	0.5519	0.25749	0.13333	0	56	6.6607	1.2287	0.24407	32
7	2.0387	0.74256	0.2585	4	58	4.651	1.0509	0.2679	36
8	0.80632	0.30093	0.18367	0	59	4.1066	0.93352	0.2373	30
9	0.53634	0.20678	0.24383	0	60	4.6111	1.0415	0.22263	36
10	2.1006	0.62061	0.23605	8	61	5.3072	1.0379	0.2224	42
11	0.90378	0.28761	0.25455	0	62	3.4576	0.86013	0.25535	36
12	1.4141	0.39255	0.21901	2	63	3.3499	0.76603	0.20896	24
13	2.7209	0.64771	0.17653	8	64	4.2558	0.97523	0.2268	38
14	2.2166	0.48488	0.17848	4	65	4.3582	0.99347	0.26681	42
15	1.8441	0.6737	0.26213	6	66	3.6229	0.86813	0.2763	34
16	1.3583	0.34007	0.19625	6	67	4.9184	0.95597	0.28307	42
17	2.9587	0.80691	0.19821	10	68	6.153	1.3444	0.27141	44
18	3.0326	0.8044	0.23974	12	69	5.9667	1.1288	0.25992	40
19	2.1888	0.49766	0.2238	6	70	5.4988	1.0149	0.26648	54
20	1.8103	0.51545	0.19723	8	71	3.4999	0.76117	0.28605	48
21	3.3897	0.85487	0.23964	8	72	3.9549	0.8523	0.20687	32
22	1.7787	0.45268	0.20753	6	73	3.8756	0.92051	0.27438	50
23	2.5515	0.52189	0.2065	14	74	5.1738	1.0749	0.24383	44
24	3.1837	0.80817	0.23931	10	75	4.6221	1.1125	0.25376	46
25	3.5666	0.71056	0.19116	18	76	4.0006	0.83618	0.26773	38
26	3.6128	0.80835	0.19037	14	77	4.6146	1.0362	0.30644	48
27	2.4562	0.65699	0.22612	14	78	3.9919	0.90495	0.27657	46
28	1.9827	0.48629	0.23574	6	79	4.505	0.92648	0.27406	40
29	3.7706	0.64217	0.25356	18	80	4.977	0.87433	0.22441	62
30	2.9596	0.71814	0.2231	16	81	6.6098	1.1362	0.21838	34
31	3.4819	0.77256	0.20048	20	82	6.0548	1.0165	0.25419	50
32	2.4687	0.56578	0.21413	22	83	4.7422	1.0693	0.28225	54
33	2.7102	0.62903	0.2193	2,2237	84	6.4159	1.1802	0.24533	58
34	3.2263	0.6624	0.24786	16	85	4.5245	0.99536	0.25921	52
35	3.3733	0.84613	0.19564	26	86	3.4366	0.83817	0.29381	38
37	3.481	0.88542	0.25249	22	87	6.3453	1.0076	0.2842	58
38	6.437	1.4839	0.19914	24	88	4.9297	1.2173	0.28597	62
39	5.0999	1.1665	0.27235	24	89	4.9228	1.0407	0.20741	48
40	4.4078	0.76662	0.17334	14	90	5.3443	1.2316	0.22649	50
41	3.6457	0.85874	0.25	24	92	4.9546	1.2078	0.27174	66
42	3.9827	0.93717	0.19735	30	93	5.1639	0.96083	0.26691	64
43	4.2411	1.0286	0.3019	30	94	5.3586	1.1253	0.23694	56
44	3.4912	0.96804	0.29853	26	95	6.6417	1.2489	0.22213	76
45	2.9602	0.82526	0.18117	32	96	6.4618	1.2794	0.22821	58
46	5.2983	0.87226	0.26068	28	97	4.34	1.002	0.28071	58
47	3.5694	0.65338	0.21881	28	98	4.1683	0.88517	0.25486	70
48	2.5891	0.64391	0.24845	36	99	5.3414	1.1351	0.26223	76
49	3.0046	0.77815	0.28214	32	100	5.6111	0.96708	0.25376	62
50	4.1534	0.95971	0.2209	32	101	6.7152	1.4455	0.23763	70
51	3.366	0.91707	0.22678	24	102	5.1087	1.1786	0.23894	70
52	4.0488	0.82832	0.27821	26	103	4.5172	1.0307	0.30495	58
53	4.1303	0.85931	0.24416	34	104	6.5995	1.3318	0.24182	66
54	4.672	0.84802	0.24867	30	105	5.3217	0.90848	0.22051	60
55	4.537	0.93272	0.26115	30					

Figure 4.12: Distance Metrics, UPGMA

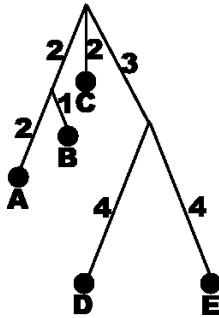


Figure 4.13: Newick Tree

between those sequences using a program like Treedist in PHYLIP and then generating our trees using those distances rather than the true distances.

Some other directions that we would like to pursue in the future are using larger trees and larger pairwise branch differences. We would also like to try a different program, like GARLI, to adjust the branch lengths on the trees produced in PHYLIP.

## 4.A Project Details

The details of how our project was carried out are given in this section.

The first task was to build the original trees in Newick format. Newick Format is a method of representing a tree with or without branch lengths as a character string. Two nodes,  $A$  and  $B$ , that share a parent are written  $(A, B)$ , and any subtrees are similarly codified, using parentheses recursively. If branch lengths are required they are included following a leaf name or parenthetical statement using a colon. So, for example, the tree  $((A : 2, B : 1) : 2, C : 2, (D : 4, E : 4) : 3)$  represents a tree on 5 nodes where  $A$  and  $B$  share a parent,  $D$  and  $E$  share a parent, and both of these parents share a parent with leaf  $C$ , as shown in Figure 4.13.

Each tree generated in this format was labeled "#newick.tree". From the newick string we produced, using "pdist" in MATLAB, a vector of pairwise distances, that is, the distances between each pair of nodes, that represented a lower triangular distance matrix, which was labeled "#distance.mtx". For example, if the newick string representing the original tree were on 3 nodes

labeled  $A$ ,  $B$  and  $C$ , then the distance vector would be of the form

$$[dist(B, A) \quad dist(C, A) \quad dist(C, B)].$$

This vector was converted to a lower triangular matrix in the format required for input into the distance matrix programs in PHYLIP. Three copies of the file were made, and the copies were renamed "#newickA", where A=F,N,U, representing the algorithm to be run on the file, Fitch-Margoliash, Neighbor-Joining, and UPGMA respectively.

Using MATLAB, we also converted the "#newick.tree" files into Dawg format. Dawg is an acronym for DNA assembly with gaps. This program takes in a tree in Newick format and returns a series of sequences assigned to each node of the tree. The sequences are generated randomly, taking into account the branch lengths, using one of several possible evolutionary models. We used the most simple, the Jukes-Cantor model. One of the output options for Dawg files is PHYLIP format. The Dawg files were saved in PHYLIP format as "#input.phl" to be used in the Dnaml program in PHYLIP.

In PHYLIP we used the programs Fitch and Neighbor to build the trees "#newickF", "#newickN", and "#newickU", where  $F$  represents that the tree was built using the Fitch-Margoliash algorithm,  $N$  the Neighbor-Joining algorithm and  $U$  the UPGMA algorithm. The programs Fitch and Neighbor take as input files some form of distance matrix, which gives the pairwise distances between the taxa of the evolutionary sequences. Fitch runs the Fitch-Margoliash algorithm, which uses a weighted least squares fit to build a phylogenetic tree. Neighbor has the option to run the Neighbor-Joining method introduced by Naruya Saitou and Masatoshi Nei in [44], which also uses a least squares method, or the UPGMA algorithm, the simplest tree building method. The "#newickF" trees were built by using the default settings of Fitch, with the exception of changing the input matrix to lower triangular. The "#newickN" trees were built by changing only the matrix setting of Neighbor to lower triangular. The "#newickU" trees were built using the UPGMA and lower triangular matrix options in Neighbor.

The newick strings of the original tree and each of the three trees built in PHYLIP were placed into a file named "#topo". The order for placement was always true, Fitch, Neighbor-Joining, UPGMA, so that tree 1 was always associated to the original tree, tree 2 to the Fitch tree, etc. These files were used as the input files for the program Treedist, which measures

the distance between two tree topologies. It will measure either the Branch Score Distance of Kuhner and Felsenstein [26] or the more widely known Symmetric Difference of Robinson and Foulds [40]. We chose to use the Robinson-Foulds distance, and compared all trees in the "#topo" file pairwise. The output from this program was chosen to be a three column table with tree  $i$  in column one compared to tree  $j$  in column two and the resulting difference between them in column three.

We also put each of the trees built in PHYLIP, together with the corresponding "#input.phl" file, through the program Dnaml. The input required by Dnaml is a set of sequences (DNA, RNA, protein). Using the sequences it can build a phylogenetic tree, determine the likelihood of a tree that was built from the sequences by another program, or adjust branch lengths on a tree built by another program. Our purpose was to adjust the branch lengths of the tree built in PHYLIP. The hope was that we would get a better fit for each tree, making it closer to the original tree. The output file from Dnaml were named "#newickA2.tree", where A=F,N,U.

At this point all of the files "#newickA.tree" and "#newickA2.tree", where A=F,N,U, were put back into MATLAB to determine their distances from the original tree by use of three different metrics.

A *metric* is a method of measurement, in our situation a tool for determining what it means for data points to be "far apart" or "close together". A *metric space* is specifically a set of objects  $A$  with a properly defined metric. We define a metric  $d : A \times A \rightarrow A$  to be a function with the following properties for all  $a, b, c \in A$ :

- $d(a, b) \geq 0$  with equality if and only if  $a = b$
- $d(a, b) = d(b, a)$
- $d(a, b) + d(b, c) \geq d(a, c)$

Once we had a vector of pairwise distances among the nodes of our trees generated by the UPGMA, Neighbor-Joining, and Fitch-Margoliash algorithms, we determined metrics by which we would compare them to the pairwise distance vector of the original tree. We settled on the metrics we call Max Distance, Mean Distance, and Precedence Distance.

By *Max distance* we mean the metric on the space of pairwise distance vectors determined by the maximum absolute coordinate-wise difference between two vectors, taken over all coordinates. In terms of phylogenetic trees this metric measures the highest absolute change in distance between two nodes after a topological change in a tree. This is also known as the  $l_\infty$  norm or the Max Norm, and it fulfills the requirements of a metric on the space of all vectors of real numbers of the same length.

By *Mean distance* we mean the metric on the space of pairwise distance vectors determined by the mean over all absolute coordinate-wise differences between two vectors. In terms of phylogenetic trees this metric measures the mean absolute change in distance between every pair of nodes after a topological change in a tree. This is also a metric, since it's a positive constant multiple (namely  $\frac{1}{n}$ ) of a known metric (the  $l_1$  or TaxiCab metric) on the space of all vectors of real numbers of the same length.

The *Precedence distance* is an attempt to measure the change of distances between two nodes after a topological change in a tree relative to the change of distances among other pairs of nodes. We first take a vector  $v = [v_1 \ v_2 \ v_3 \ \dots \ v_{\binom{N}{2}}]$  of all pairwise distances between nodes. We then create a new vector  $u$  of the same length consisting of the indices of the entries in  $v$  in ascending order. The vector  $u$  is called the *index vector*. For example, the index vector of  $[2.4 \ 5 \ 0.2 \ \pi]$  is  $[3 \ 1 \ 4 \ 2]$ , since sorting the entries of  $v$  results in the third entry taken first, then the first entry, followed by the fourth entry, and ending with the second entry. Using the index vector we generate a  $k \times k$  matrix  $P$ , where  $k$  is the length of both  $v$  and  $u$ , with entries from  $\{0, 1\}$ . We assign  $P_{ij}$  a 1 if  $i$  precedes  $j$  in the index vector and a 0 otherwise. We call  $P$  the *precedence matrix* of the tree, and it tells us the relative distances between various pairs of nodes in a tree. Note that there is a one-to-one correspondence between the set of all index vectors of length  $k$  and the set of all  $k \times k$  precedence matrices.

To determine the precedence distance between two trees, we generate each tree's precedence matrix and count the total number of positions for which the two matrices differ. Since the size of the matrix increases at a rate proportional to the fourth power of the number of nodes we must scale this value. For this reason we let the precedence distance be equal to the ratio of the value we determined above to the total number of entries in the precedence matrix.

Since absolute distance and the set of all  $n \times n$  matrices over the reals is a metric space, the set of all precedence matrices is a subset of this set of matrices, and the fact that a positive constant multiplier to one metric results in another metric, we have that precedence distance is a valid metric on the space of all index vectors. Note that it's not, however, a valid metric on the space of all trees, since two unique trees may generate the same index vector, and thus the same precedence matrix, resulting in a precedence distance of 0. This has no effect on our measurements, as we do not need a formal metric, only a valid form of measure.





# Chapter 5

## Mathematical Model for the Mutation Rate of the Avian Flu Virus

By Jennifer Reinert, Shelley Speiss, Minjeong Kim and Marcela Kuzmiak

### Abstract

The purpose of our project was to find model parameters that can be used to create a model for the mutation rate of the gene encoding for the neuraminidase in the DNA of the H5N1 virus, known as the Avian Flu virus. CLUSTALW was ran to align a set of given DNA sequences and find a distance matrix for this set. This matrix was used in the phylogenetic algorithms UPGMA and Neighboring Joining to create phylogenetic trees, which were seeded into GARLI to infer branch lengths and produce two maximum likelihood trees. From these trees, a base frequency vector and rate matrices were produced by GARLI which can be used to model the rate of mutation of this particular gene.

### 5.1 Introduction

The Avian Flu virus, H5N1, has the potential to mutate into a form that is as infectious as the 1918 H1N1 influenza pandemic which killed approximately 10% of the world population in as little as six months. To date, there are no

vaccines for this virus. In the event that H5N1 mutated into a form infectious to humans, the use of a vaccine would be inefficient to stop and treat such a pandemic since it takes at least six months to develop and produce vaccines. There are, however, treatments using therapeutic agents. One such agent is oseltamivir (under the brand name "Tamiflu"), which is designed to inhibit the neuraminidases, the structure of the virus that aids the transmission of the virus from cell to cell. The H5N1 neuraminidase genotypes have a wide range of responses to oseltamivir, which is presumed to be caused by the variations of neuraminidase structure. These variations are due to mutations in the specific gene in the virus' DNA encoding for neuraminidase. As the mutations continue to occur in this gene, the effectiveness of the oseltamivir and similar agents may diminish. The problem to study is to predict how the gene will mutate in order to develop better drug treatments to prevent a potential pandemic.

Jack Horner with the SAIC presented our clinic class with the problem of finding a way to model the rate of mutation for the gene encoding for the neuraminidase of the H5N1 virus and produce evaluations of the different steps involved in finding this model. The particular task assigned to our group was to take the Los Alamos National Laboratory influenza database provided by Jack Horner and find the parameters needed to create the model. [30] The flow of our work is illustrated below:

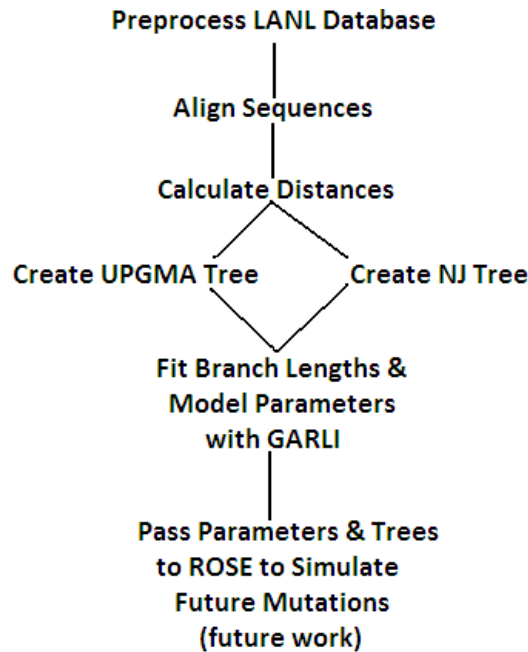


Figure 5.1: Flowchart of the project.

The first step in the project, to preprocess the dataset, was done in order to create a reliable source of data. The original data set consisted of approximately 1200 DNA sequences of various lengths. Of that set, about 8% of the sequences contained less than a 1000 base pairs, i.e. the length of the sequence was less than 1000. Jack Horner requested that these particular sequences be removed, as they might lead to problems in finding accurate parameters for the model. Once this process was done, the data set was reduced to 1102 sequences. The "cleaned-up" data set was then aligned. Alignment of the DNA sequences attempts to find similarities between the base pairs within the sequences, with gaps inserted where similarities cannot be found. Once aligned, a distance matrix can be created between pairs of sequences to express how closely related the sequences may be. The program CLUSTALW was used to both align and create a distance matrix for the given data set. This part of the project was done by Jeff Kenyon, another member in our clinic class.

The next step involved is finding the phylogenetic trees to use. There are a number of different phylogenetic algorithms that create a tree topology

indicating the evolutionary relationship among the DNA sequences. Phylogenetic trees can be "seeded" into programs such as GARLI, i.e. these programs can use pre-existing trees to improve upon, to find model parameters. Seeding with trees may speed up the computation time in finding parameters, effect the accuracy of the parameters given, or show discrepancies in the data set. We chose to use the UPGMA and Neighbor Joining algorithms to create the phylogenetic trees to seed GARLI with, because they are both readily available in the bioinformatics toolbox in MATLAB and have a quick computation time. These algorithms both require the distance matrix found using CLUSTALW to create the phylogenetic trees. Also considered was creating a tree using the GARLI program, which does not require the distance matrix. We decided against this in the end, since the run time on finding model parameters with a tree topology from GARLI was longer than that of using a tree topology from UPGMA or Neighbor Joining.

Having found the tree topologies we wanted to use, these were then passed into GARLI to fit branch lengths to the phylogenetic trees using maximum likelihood methods. Both the tree topologies and the branch lengths can be modified and GARLI does allow for some constraining of the tree topology itself, as described in section 2.3.2. For our project we did not add in any constraints to the starting tree topologies we seeded GARLI with. After finding branch lengths of these non-fixed tree topologies, GARLI can produce a base frequency vector (the frequency of the characters A, G, T, and C appearing in the sequences) and a rate matrix with entries describing the rate at which one character may mutate into another character (or vice versa). Additional output from GARLI is a maximum likelihood tree created from the original trees that were used in seeding. The maximum likelihood tree, the base frequency vector, and the rate matrix are required for producing a model in either GARLI or the program ROSE.

Within our report, we describe the algorithms UPGMA and Neighbor Joining, as well as the program GARLI. Under the results section will appear the phylogenetic trees created using UPGMA and Neighbor Joining, the maximum likelihood trees created from these using GARLI, and finally the model parameters (base frequency vector and rate matrices) found to create a model.

## 5.2 Background

### 5.2.1 UPGMA - Unweighted Pair Group Method with Arithmetic Mean

#### Introduction

UPGMA is one of the simplest methods of phylogenetic tree construction [35]. Its simplicity comes from assuming a constant rate of mutation. It uses a sequential clustering algorithm where it identifies the smallest distance between two operational taxonomic units (OTUs or taxons) and then treats them as a new single composite OTU. It continues by finding the smallest distance from the OTUs in this new group, and repeating the process until only two taxons remain and a tree is formed.

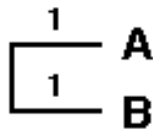
#### Algorithm

Seeing UPGMA is the best way to understand this simple algorithm. The following example comes from Opperdoes [35].

Suppose we have a distance matrix:

	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
<i>B</i>	2				
<i>C</i>	4	4			
<i>D</i>	6	6	6		
<i>E</i>	6	6	6	6	
<i>F</i>	8	8	8	8	8

The smallest distance is between *A* and *B*. We divide their distance by 2 and get  $2/2 = 1$  and we construct a subtree:



This becomes a new composite OTU(*A, B*) and we calculate a new dis-

tance matrix:

$$\text{dist}[(A, B), C] = [\text{dist}(A, C) + \text{dist}(B, C)]/2 = (4 + 4)/2 = 4$$

$$\text{dist}[(A, B), D] = [\text{dist}(A, D) + \text{dist}(B, D)]/2 = (6 + 6)/2 = 6$$

$$\text{dist}[(A, B), E] = [\text{dist}(A, E) + \text{dist}(B, E)]/2 = (6 + 6)/2 = 6$$

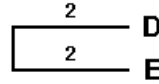
$$\text{dist}[(A, B), F] = [\text{dist}(A, F) + \text{dist}(B, F)]/2 = (8 + 8)/2 = 8$$

where the distance between a simple OTU and a composite OTU is the average of the distances of the components of the composite and the simple OTU.

We get the new matrix:

	<i>A, B</i>	<i>C</i>	<i>D</i>	<i>E</i>
<i>C</i>	<b>4</b>			
<i>D</i>	6	6		
<i>E</i>	6	6	4	
<i>F</i>	8	8	8	8

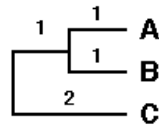
And a new subtree:



And construct a new distance matrix:

	<i>A, B</i>	<i>C</i>	<i>D, E</i>
<i>C</i>	<b>4</b>		
<i>D, E</i>	6	6	
<i>F</i>	8	8	8

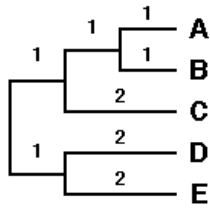
And a new subtree:



Then the new distance matrix:

	<i>AB, C</i>	<i>D, E</i>
<i>D, E</i>	<b>6</b>	
<i>F</i>	8	8

And a new subtree:



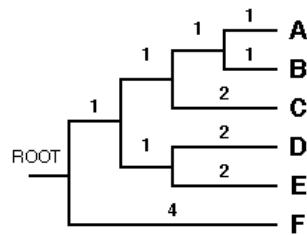
In the final step we have the last clustering:

	$ABC, DE$
$F$	8

This gives us a distance to the root as:

$$\text{dist}[(ABCDE), F] = 8/2 = 4$$

And the tree given by UPGMA is:



## Conclusion

The tree produced by UPGMA is ultrametric, i.e. it shows the leaves as equidistant from the root. It means that all offspring we have mapped have evolved an equal distance from their parent. To illustrate, consider the example given above. The common ancestor of  $A$ ,  $B$ ,  $C$ ,  $D$ , and  $E$  is 3 “ticks” from each. This is because the algorithm assumes a constant mutation rate. Also, the complexity of UPGMA is  $O(n^2)$ , since there are  $n - 1$  iterations, with  $O(n)$  work done in each iteration. While UPGMA runs very quickly, this algorithm may not produce an accurate tree topology, especially when compared to the accuracy of the following algorithm, Neighbor Joining which also runs quickly but also produces a tree topology that is close to what the true topology might be.

## 5.2.2 Neighbor Joining

### Introduction

Minimum evolution is the main principle of constructing a phylogenetic tree. The minimum evolution method used in the construction of phylogenetic trees is based around the assumption that the tree with the minimum branch length sum is most likely to be the true tree. The standard method producing minimum evolutionary trees examines all possible tree topologies or a given number of them, and finds the tree with the least amount of evolutionary change and names that tree as the final one. The evolutionary change is measured by how much the final tree has changed from its parent tree. Like many other methods for constructing phylogenetic trees, Neighbor Joining uses a distance method that proves to be most efficient. This greedy algorithm is efficient because it explores only a small part of the solution space but because of that, the best solution may be missed. Like many other algorithms of its kind, Neighbor Joining “does not necessarily produce the minimum-evolution tree, but computer simulations have shown that it is quite efficient in obtaining the correct tree topology” [32]. Never the less, Neighbor Joining has proved to be “quite efficient compared to other tree-making methods that produce a single parsimonious tree” [32].

### History

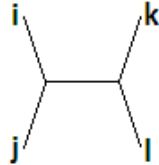
Saitou and Nei created the Neighbor Joining Method in 1987 and it has since been modified by Studier and Kepler in 1988 [32].

### Algorithm

The basis behind Neighbor Joining is the 4-Point Condition. The 4-Point Condition is an inequality for neighbors that states that for any four leaves on a tree,  $d(ij) + d(kl) < d(ik) + d(jl)$  and  $d(ik) + d(jl) = d(il) + d(jk)$  where  $i, j, k$  and  $l$  are leaves on the tree and  $d$  = the sum of the edges joining the two points. The center edge joining  $ij$  and  $kl$  is not taken into account in the first quantity and is taken into account twice in the second and third quantity [53]. More simply stated, the 4-Point Condition states that the distance between  $i$  and  $j$  plus the distance between  $k$  and  $l$  must be less then the distance between  $i$  and  $k$  plus the distance between  $j$  and  $l$  because in



the later, the middle branch is accounted for twice when in the former it is not accounted for at all.

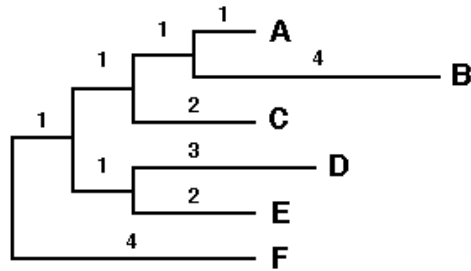


The Neighbor Joining method starts out with a pair of OTUs which are connected through a single node and labeled as neighbors. From here the distance can be calculated between the first set of neighbors and each other node on the tree. The distance can be calculated by taking the sum of the branch lengths separating the two nodes. Once the distance is calculated, the smallest distance sum between nodes becomes the new neighbors. This process is continued until there are only two nodes remaining that are connected by a single branch.

An example of how the Neighbor Joining is run on a given set of data follows:

Start with the following example:

Given this tree:



a matrix can be made to represent the distance between each of the six OUT's ( $N = 6$ ).

	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
<i>B</i>	5				
<i>C</i>	4	7			
<i>D</i>	7	10	7		
<i>E</i>	6	9	6	5	
<i>F</i>	8	11	8	9	8

The first step in this process is to calculate for each OTU the sum of the distances between it and the other OTUs on the tree.

$r(i) = (\text{distance from } i \text{ to } A) + (\text{distance from } i \text{ to } C) + \dots + (\text{distance from } i \text{ to } X)$ .

For example,

$$\begin{aligned} r(A) &= 5 + 4 + 7 + 6 + 8 = 30 \\ r(B) &= 42 \\ r(C) &= 32 \\ r(D) &= 38 \\ r(E) &= 34 \\ r(F) &= 44 \end{aligned}$$

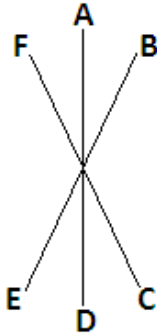
Given our new distances we can now produce a new matrix calculating the distance from each pair of OTU's by:

$$\begin{aligned} M(ij) &= d(ij) - [r(i) + r(j)] / (N - 2) \\ M(AB) &= (5) - [(30) + (42)] / (6 - 2) = 5 - [72] / (4) = 5 - [18] = -13 \\ M(BC) &= (7) - [(42) + (32)] / (6 - 2) = 7 - [74] / (4) = 7 - 18.5 = -11.5 \end{aligned}$$

Going back to the 4-Point Condition, the new distance is calculated by taking the distance from  $i$  to  $j$  and then adding the sum of their individual distances divided by the number of OTU's over 2. Continue this process until the following matrix is produced. This matrix is the new distance matrix produced when the above step is performed.

	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
<i>B</i>	-13				
<i>C</i>	-11.5	-11.5			
<i>D</i>	-10	-10	-10.5		
<i>E</i>	-10	-10	-10.5	-13	
<i>F</i>	-10.5	-10.5	-11	-11.5	-11.5

Neighbor Joining always starts with a star like tree, produced by assuming there is no clustering of OTU's.



From the new matrix produced, we select the smallest distance between OTU's as neighbors. The new pair of OTU's are combined as a single OTU defined as  $U$ . The smallest distance is  $-13$  which is both  $A$  and  $B$  and  $C$  and  $D$ . Select  $A$  and  $B$  for this example. Then we calculate the distance from  $A$  to  $U$  and  $B$  to  $U$  by:

$$\begin{aligned}
 S(AU) &= d(AB)/2 + [r(A) - r(B)]/2(N - 2) \\
 &= (5)/2 + [30 - 42]/2(6 - 2) = 5/2 + [-12]/(2 \times 4) \\
 &= 5/2 + [-12/8] = 5/2 + [-3/2] = 1 \\
 S(BU) &= d(AB) - S(AU) = (5) - (1) = 4
 \end{aligned}$$

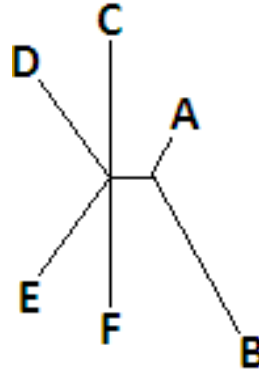
Once the first pair of neighbors have been selected, a new matrix must be created with  $A - B$  defined at one OTU defined as  $U$ . The distance from  $U$  to each other OTU must again be calculated.

$$\begin{aligned}
 d(CU) &= d(AC) + d(BC) - d(AB)/2 = 4 + 7 - 5/2 = 6/2 = 3 \\
 d(DU) &= d(AD) + d(BD) - d(AB)/2 = 6 \\
 d(EU) &= d(AE) + d(BE) - d(AB)/2 = 5 \\
 d(FU) &= d(AF) + d(BF) - d(AB)/2 = 7
 \end{aligned}$$

Then another matrix is created with  $A - B$  as one OTU:

	$U$	$C$	$D$	$E$
$C$	3			
$D$	6	7		
$E$	5	6	5	
$F$	7	8	9	8

The OTU's *A* and *B* are then split off from the star and paired as a single OTU reducing the number of OTU's to 5. The new tree looks like:



This process is repeated until there are only  $N = 2$  OTU's remaining. The resulting tree is a binary tree [36].

### Efficiency

Neighbor Joining is very inexpensive in terms of the time it takes to run the algorithm. But is the reduction in cost more valuable than the accuracy it loses? On very small sets of sequences, Neighbor Joining is likely to produce the exact optimal minimum evolutionary tree [38]. But when the number of sequences that Neighbor Joining receives becomes large, is it still efficient enough to run? The answer is yes. According to a study done by Koichiro Tamura, Masatoshi Nei, and Sudhir Kumar, when the number of sequences is increased to hundreds or even thousands, the accuracy does not seem to change much at all. The accuracy is “measured by the percentage of phylogenetic clades (a group of organisms comprised of a common ancestor and all of its descendants) correctly inferred”. In the study they ran, they found that when they increased the sequence value  $m$  from  $m = 32$  to  $m = 4,096$ , there was only a 2.2% decrease in the accuracy.[28]

### Comparison

There are many other distance based methods that can be used to quickly produce phylogenetic trees, some with similar methods to Neighbor Joining and some with very different methods. The Sattath and Teversky method counts the number of cases that satisfy the 4-Point Condition for each pair of

OTU's and choose the pair that have the largest value as neighbors. Fitch, on the other hand, uses interior-distance matrices to construct topologies and UPGMA uses an average distance method.[32] Naruya Saitou and Masatoshi Nei ran a simulation to test the efficiency of six distance method algorithms including UPGMA, Neighbor Joining, Fitch and other distance method algorithms. Each of the algorithms tested produces a single parsimonious tree from a distance matrix. They compared reconstructed trees with their model trees. When this was done, they found that UPGMA was the poorest performer and Neighbor Joining was neck in neck with the Sattath and Tversky method.[32]

### **Advantages and Disadvantages**

Neighbor Joining is very time efficient and accurate on large and small sets of data. It also allows lineages with very different branch lengths. On the other hand, there are a few disadvantages like with any algorithm. Neighbor Joining only looks at a few of the solutions and so it may end up not finding the best minimum evolutionary tree. The algorithm is also strongly dependent on the model of evolution that is used and it produces only one tree topology [36].

### **Negative Branch Lengths**

When running the Neighbor Joining algorithm it is possible to get negative branch lengths. There are two options when this happens. The first way to eliminate the negative branch lengths is to assume that every branch length must be positive and any branch length that does turn out negative is then changed to zero. The other way is to assume that negative values are due to an error in the sample and then take the absolute value of each negative number.

### **Programs**

There are different compute programs to run Neighbor Joining and all will produce the exact same tree. For the purpose of the Math Clinic, we used the `seqneighjoin` function from MATLAB in the Bioinformatics toolbox. The PHYLIP package also runs Neighbor Joining, along with UPGMA, Fitch and many other algorithms used to produce phylogenetic trees.

## Conclusion

As minimum evolutionary model algorithms go, Neighbor Joining is one of the top algorithms. Between the cost efficiency and the accuracy to produce the correct topology, there are not many that can compare. Although it may not always produce the minimum evolutionary tree desired, it should be noted that the real minimum evolutionary tree is not always a true tree either. Nonetheless, trees produced by algorithms such as Neighbor Joining and UPGMA can be used in programs such as GARLI to find the model parameters that we are looking for. It should be noted that seeding GARLI with a tree topology cuts down the computational time of producing the parameters, which played a big role in our decision to use phylogenetic trees produced by UPGMA and Neighbor Joining as starting topologies in GARLI.

### 5.2.3 GARLI - Genetic Algorithm for Rapid Likelihood Inference

#### Introduction

The program GARLI performs phylogenetic inference on given aligned nucleotide dataset under the maximum likelihood criterion. Given a set of aligned nucleotide sequences, GARLI finds high quality solutions which represent an evolutionary relationship between sequences. Free software is available at:

<http://www.bio.utexas.edu/faculty/antisense/garli/Garli.html>.

This software provides both a serial algorithm and a parallel MPI version. The GARLI version 0.95 manual has a sample dataset and a configuration file. The execution of Garli follows the flow chart shown in Figure 5.2.

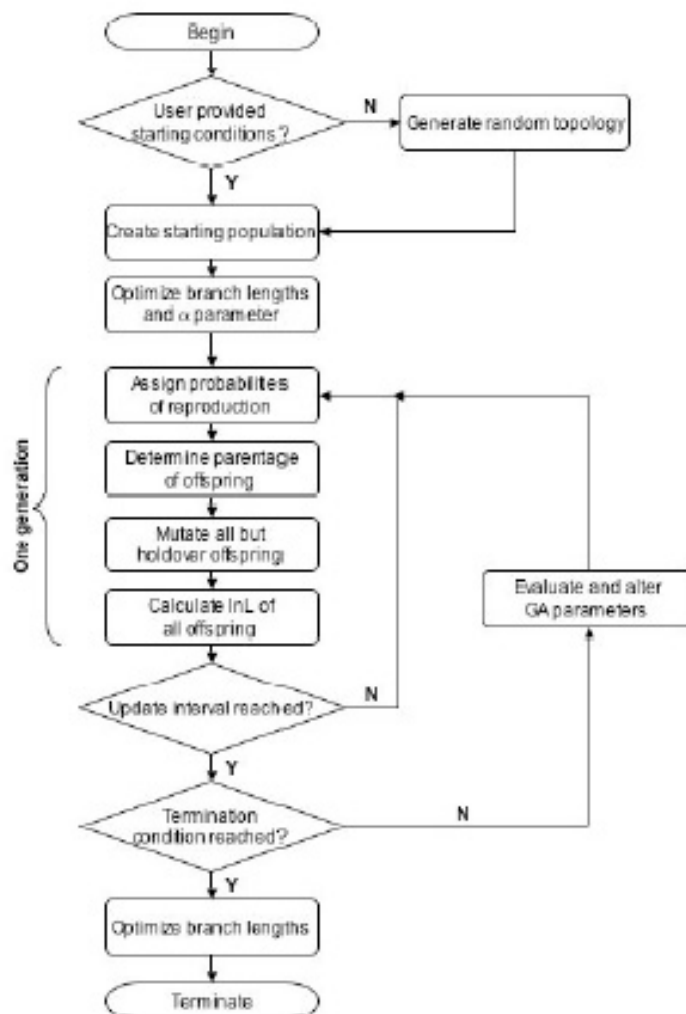


Figure 5.2: GARLI flow chart.

The GARLI software was developed to specify starting topology and model specification settings, such as 6 or 3 relative rates of nucleotide frequencies, base frequencies, and a model of the rate heterogeneity for example. Setting model parameters allows us to do some experiments and find the best tree topologies and output model parameters when our model was unclear.

Setting a starting tree topology allows us to check how the starting topology effects the computation time or model parameters. Constraining the phylogenetic tree topology is one of specifications that can be done by setting a constraint format. Changing the number of generations and modifying stopping criteria are also available. Based on the manual, several experiments can be done with this software.

### Implementation of GARLI

In our project, we focused on getting model parameters using 6 relative rates which can be used by ROSE for the final mutation model. We wanted to compare the model parameters obtained by using specified starting topologies from Neighbor Joining and UPGMA trees and using constrained tree topologies. Unfortunately, we were only able to use tree topologies without constraints. To use the phylogenetic trees produced from UPGMA and Neighbor Joining, a couple of extra steps were required. GARLI only takes the starting tree topology input in the form of Newick Standard tree, which is used for representing trees in a computer-readable format, by making use of the correspondence between trees and nested parentheses, as a starting tree topology [54]. Thus, the created file types from MATLAB need to be in Newick file format. In addition, the created output tree files from MATLAB contained characters which GARLI could not read, so that removing these characters from tree files was also required.

Fixing a tree topology is not possible since GARLI is taking Newick Standard tree as a starting tree topology. It is well known that the Newick Standard tree does not make a unique representation of a tree, for two reasons [54]. First, the left-right order of descendants of a node affects the representation, even though it is biologically uninteresting. Thus

$$(A,(B,C),D);$$

is the same tree as

$$(A,(C,B),D);$$

Moreover, GARLI allows us to put a '+' sign to group nodes only at one place at the beginning of constraint file format. So we find that it is impossible to fix tree topologies from Neighbor Joining and UPGMA trees entirely.



Learning how to constrain a tree topology in certain places required some experiments with a small dataset to see how it works. To start, we needed to remove any branch lengths from Newick file format so that GARLI could read the file. We did some experiments with given dataset and with the following experimented constraint obtained the below outputs and observations:

1. Constraint is the same as the starting tree :

- Constraint file :

```
+((40,((43,42),41)),(((32,33),((34,35),((38,(37,39)),36))),((((31,29),(30,28)),27),((8,9),(12,(10,(11,((14,13),((16,17),15))))))),((2,(3,4)),((6,7),5)),1))),((25,26),((24,(21,22)),23)),((19,20),18))),((61,52),((((55,54),56),(59,(57,58))),((62,(44,((47,(48,46))),49),(64,63))),50)),(60,(51,53))),45));
```

- Output tree:

```
(19,((18,((((17,16),(10,((12,11),(15,(14,13))))),((9,8),(1,((5,(6,7)),((4,3),2)))))),((25,26),((24,(21,22)),23)),(27,((28,30),(31,29))),((((56,(55,54)),((((63,64),(49,(47,(46,48))))),62),((34,(35,((38,(39,37)),36))),32,33),(((42,43),41),40))),44),50)),(59,(58,57)),(45,60)),((53,51),(61,52))))),20);
```

→ GARLI can only fix part of tree topology, like leaves or branches.

2. Grouping (32,33,34,35,36,37,38,39) and (40,41,42,43) with random start tree.

- Constraint file:

```
+1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,(32,33,34,35,36,37,38,39),(40,41,42,43),44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64);
```

- Output tree:

```
(((32,33),((34,35),((38,(37,39)),36))),((40,((43,42),41)),((61,52),(((55,54),56),(59,(57,58))),((62,(44,((47,(48,46))),49),(64,63))))),50)),(60,(51,53))))),45)),((((31,29),(30,28)),27),(((8,9),(12,(10,(11,((14,13),((16,17),15))))))),((2,(3,4)),((6,7),5)),1))),((25,26),((24,(21,22)),23)),((19,20),18))));
```

→ Grouped (40,41,42,43) but it cannot be unique.

3. Group (40,((43,42),41)):

- Constraint file:

```
+(+(40,+(+(43,42),41)),(((32,33),((34,35),((38,(37,39)),36))),((((31,29),(30,28)),27),(((8,9),(12,(10,(11,((14,13),((16,17),15))))))),((2,(3,4)),((6,7),5)),1))),((25,26),((24,(21,22)),23)),((19,20),18))),(((61,52),(((55,54),56),(59,(57,58))),((62,(44,((47,(48,46))),49),(64,63))))),50)),(60,(51,53))))),45));
```

- Output tree: none.

→ The GARLI software could not read this format file. It did not give us any error message but the software tried to read the format file for ten minutes, before we terminated the program. Taking this much time in reading the format file did not make sense to us. Thus, this experiment tells us GARLI only allows us to constrain some portion of the tree topology, but not all of the tree. Without a '+' sign on at the beginning of the constraint, the format file would not work; for example a constraint file such as '(+(40,+(+(43,42),41)),(((32,33),\*\*\*\*\* \*\*))';.

## Conclusion

Overall the process was automated to perform GARLI by using MATLAB, where we could create the UPGMA and Neighbor Joining trees in MATLAB and run these trees as seeds in GARLI to create the model parameters. This process is saved as the file grp2-go\_garli.m.

## 5.3 Results

Results provided are the phylogenetic trees found and the parameters produced by each tree. Some observations are provided, with the intent of informing the reader of possible areas of future work. When the term subtree is used, it is meant to imply the smaller trees formed off the first branchings off the initial node in the trees.

### 5.3.1 UPGMA and Neighbor Joining Trees

It was suggested at the beginning of the term to create two or three different phylogenetic trees to create different models that could be compared, based on the fact that whatever tree used and the model found thereafter, a single model might not accurately depict the mutation of the gene. So by using two different trees to produce two different sets of model parameters, we can compare the output of the model and hopefully find a common trends in both. Also by using more than one tree, any oddities that could arise in the topologies or in the parameters could point out problems with the data set or possibly with the methods used to find the tree or the parameters.

The first phylogenetic tree to be presented, in Figure 5.3, is the tree obtained from running UPGMA. What is interesting to note is the two long beginning branch lengths, before any new subtrees are formed. On the top branch is where most of the sequences can be seen, while on the bottom branch there are only a few. This would imply that the evolutionary distance between sequences on the top branch to sequences on the bottom branch would be great, at least according to UPGMA.

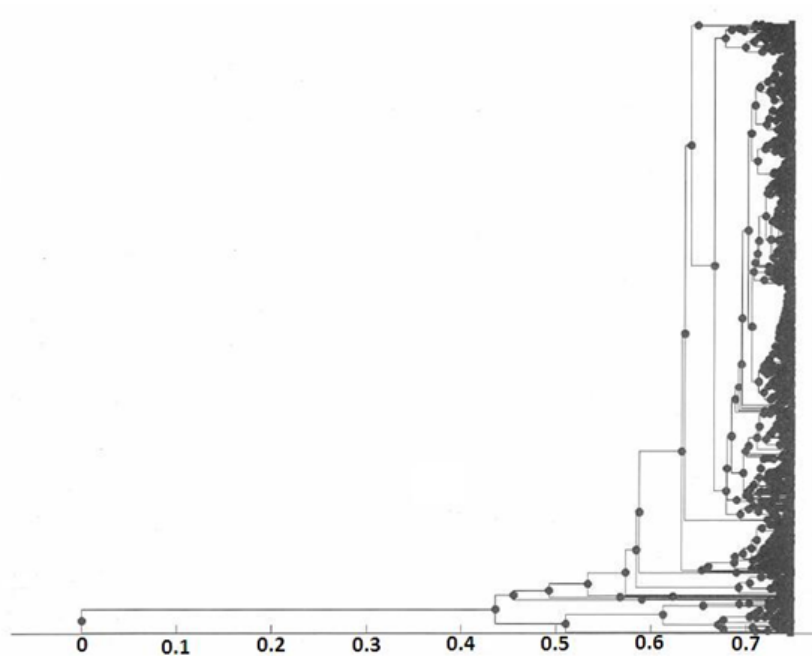


Figure 5.3: Tree created by UPGMA.

Our second phylogenetic tree shown in Figure 5.4 is the tree produced by running Neighbor Joining. Again, there are sequences appearing further away from a majority of the other sequences. It has been speculated that these sequences that are further from the general clustering of sequences could be the same sequences that are seen on the bottom branch referred to in the UPGMA tree. The only difference is the sequences in the Neighbor Joining tree that have this longer branch length are adjacent to clusters of sequences, and are not residing on their own subtree.

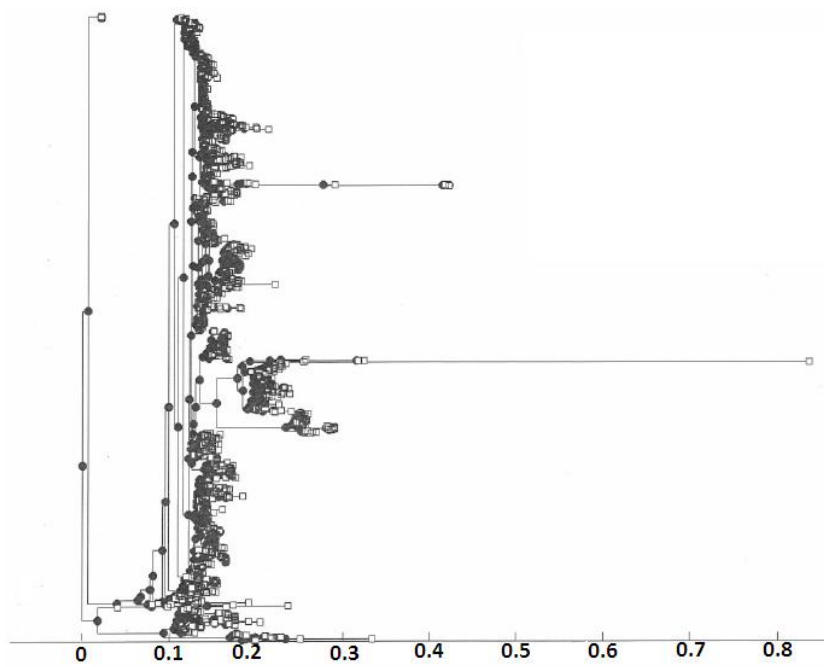


Figure 5.4: Tree created with Neighbor Joining.

### 5.3.2 Maximum Likelihood Trees

The next set of phylogenetic trees were created by GARLI by using maximum likelihood methods, with the UPGMA and Neighbor Joining trees as seeds. When reviewing these trees with Jack Horner, we see that the normal mutation of DNA sequences is accurately depicted in the trees, i.e. there are no longer sequences seen to be apart from the main clustering of sequences, and we see several subtrees formed.

The phylogenetic tree obtained from the UPGMA tree is shown in Figure 5.5. Interesting to note is there appears to be four subtrees formed from the initial starting node of the tree, as well as more branching occurring earlier on.

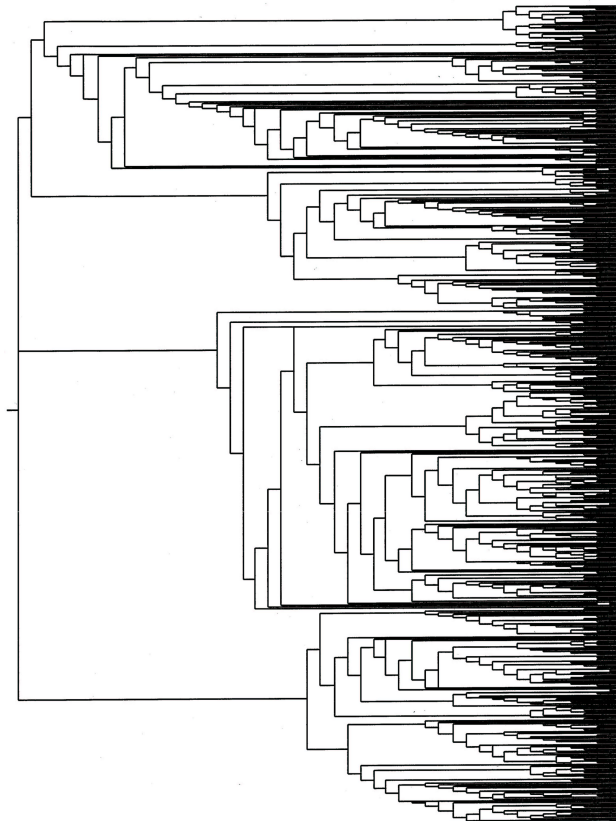


Figure 5.5: Tree created with UPGMA tree seed.

The phylogenetic tree obtained from the Neighbor Joining tree is given in Figure 5.6. Again more branching occurring earlier on is seen when compared to the original Neighbor Joining tree. There also appears to be four or five different subtrees formed.

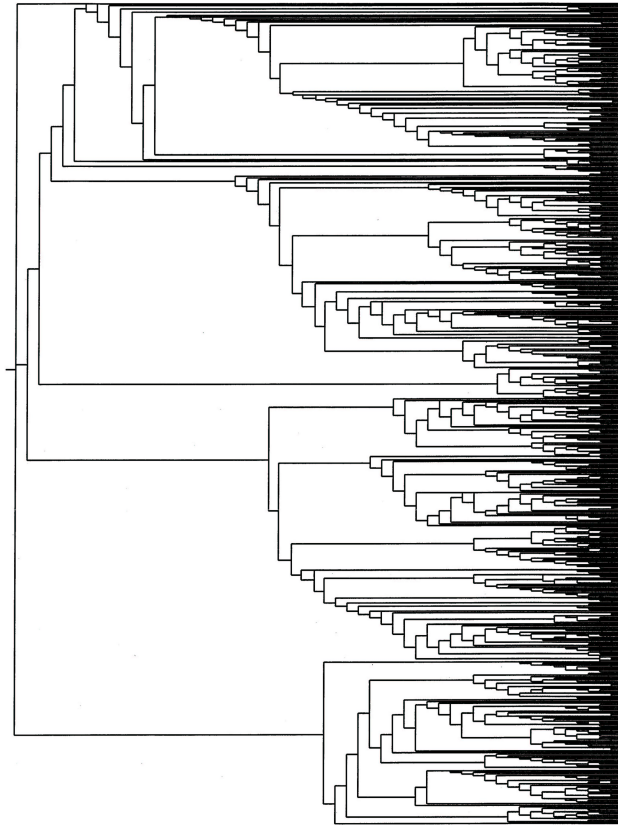


Figure 5.6: Tree created with Neighbor Joining tree seed.

### 5.3.3 Parameters

From the Maximum Likelihood trees, GARLI was able to produce the following base frequency vector (which is the same no matter which tree GARLI is seeded with), where the entries are approximated:

$$\pi = ( 0.31 \quad 0.2 \quad 0.24 \quad 0.25 )$$

The following rate matrix, shown with approximate entries, was produced when the UPGMA tree was used as a seed:

$$\mathbf{R} = \begin{pmatrix} 2.09 & 6.86 & 0.96 \\ & 0.62 & 10.01 \\ & & 1 \end{pmatrix}$$

The rate matrix obtained from GARLI with the Neighbor Joining seed as a tree, also with approximate entries, is similar to the above matrix:

$$\mathbf{R} = \begin{pmatrix} 1.82 & 6.06 & 0.97 \\ & 0.57 & 8.33 \\ & & 1 \end{pmatrix}$$

Of particular interest when looking at these matrices are the high values of 6.86 and 6.06 and the even higher values of 10.01 and 8.33 occurring in the same positions. When talking to Jack Horner about the matrices, he suggested that this could possibly have something to do with the data set itself, or with the way the algorithms produce the phylogenetic trees. A way to investigate this anomaly would be to seed GARLI with different phylogenetic tree, perhaps produced by an algorithm using different techniques than that of UPGMA and Neighbor Joining, and compare the rate matrix obtained from this tree. If this approach proved unfruitful, another idea would be to consider the data set itself and possibly incorporate the tree topologies found as guides as to which sequences to look into that may be affecting the rate matrix.

## 5.4 Conclusion and Future Work

From the start of the class project, the aim was to find a model to see how the gene encoding for the H5N1 neuraminidase would mutate. Through our work, we've set the grounds for how to go about finding the parameters to



produce this model. Though there are a number of phylogenetic algorithms to use, and different programs besides GARLI to then create the parameters, we have illustrated the means needed to produce parameters for evolutionary models using readily available resources. The next step would be to use a program such as ROSE to create the final product, the model of the mutation rates. This last step was the final goal of our group, but because of time we fell short of this goal. And so creating the model of the mutation rates would definitely fall under the title of "future work." But with the results we were able to gain, from the tree topologies to the interesting rate matrices, several different avenues opened up that could also be considered future works.

One such topic would be comparing a phylogenetic tree created in GARLI to the original phylogenetic tree used to seed GARLI with, for example comparing the UPGMA tree to the maximum likelihood tree created from using the UPGMA tree as a seed in GARLI. Possible ways to analyze the problem could include graph theoretic approaches, comparing fitted branch lengths, or examining the sequences themselves and how they are related in each topology. There is also comparing phylogenetic trees obtained from other algorithms to UPGMA and Neighbor Joining tree topologies, both before and after seeding in GARLI. These kinds of steps may help provide clues to the variations in the entries of the rate matrices, which is another topic to be considered.

After studying the phylogenetic trees given, there may exist certain sections of these trees where grouping certain sets of leaves (sequences) may alter the entries found in the rate matrices. The technique to constrain groups of leaves is outlined in section 2.3.2. It would be interesting to see how much the rate matrix can change depending on how many leaves and which leaves are constrained.

There is also coming up with several different models of the mutation rate and comparing them for anything that may seem strange or perhaps normal. We've already provided the parameters for two, but without a third or fourth set of parameters to use, it is hard to tell if the unusual entries in these rate matrices have something to do with using UPGMA and Neighbor Joining in creating the original trees or the dataset itself, and the effect these variations will have on the mutation model. Originally we hoped to have used a phylogenetic tree created by GARLI for a third comparison, but because of the run time we were unable to, so there is at least a third set of parameters out there to be found for comparison. In the end, creating a data base of trees and model parameters was our goal, so that such comparisons could be

possible. While we may not have achieved this goal, we certainly proved it was possible and provided an example of how the process works and showed the results that are obtainable.

## Chapter 6

# Performing Multiple Sequence Alignment on Very Large Data Sets

By Jeff Kenyon

### Abstract

CLUSTALW has been an established standard in multiple sequence alignment for many years. Recently, other programs have been introduced that use different techniques to perform multiple sequence alignment with higher accuracy and speed. This paper discusses the various approaches to multiple sequence alignment embodied by these programs, and uses simulated data sets to evaluate the speed and accuracy of these programs. The resulting alignments are then used as input to recreate the phylogenetic tree. This study found that for large sequence sets, MAFFT, using the NS-i algorithm, is able to perform a more accurate multiple sequence alignment than CLUSTALW, in less time. However, the study was inconclusive with regard to the hypothesis that a more accurate multiple sequence alignment results in a more accurate phylogenetic tree.

## 6.1 Introduction

The process of multiple sequence alignment is an NP-complete problem [6], and is performed using heuristic methods. The CLUSTAL family of programs were considered the standard, but in recent years, new approaches have been introduced that are both faster and more accurate than CLUSTALW. To address the current task of creating an MSA for the 1000+ sequences in the LANL data set, this project addresses whether one of the newer generation of alignment programs is more suitable than CLUSTALW. This question will be examined by comparing generated MSAs against a known, correct MSA from a simulated data set, using an accepted metric.

This project further seeks to investigate whether a more accurate MSA results in a phylogenetic tree that can be assessed as more accurate (using the same metrics as were used in the clinic project evaluating phylogenetic tree generation programs). This question will be examined by using the MSAs to generate phylogenetic trees, and then comparing them to the known, correct evolutionary tree created as part of the generated data set.

This project presents the results of the experiments outlined above, with the conclusion that, while MAFFT (NS-i) can achieve a better result than CLUSTALW in less time, the impact of the more accurate MSA upon phylogenetic tree generation is unclear. This lack of clarity would require significant further study to unravel, as it may involve several factors, including the use of custom generated data sets, rather than benchmark data sets, as well as the choice of software for generating the phylogenetic trees.

## 6.2 Multiple Sequence Alignment

Multiple Sequence Alignment (MSA) is the process of using multiple sequences to form a single “consensus” sequence representative of the entire group. The individual sequences are then aligned (and gaps hypothesized) against this theoretical consensus alignment [6]. The alignment process is a means of identifying homologous segments between sequences.

Although the MSAs produced in this project were intended as the basis for phylogenetic tree generation, an MSA is interesting in its own right, as it may help identify functional areas of the gene; nucleotide chains that show little or no change from sequence to sequence, known as *highly conserved regions*, are typically crucial to the gene’s function [6].

There are several basic approaches to the alignment task. The most common are progressive alignment, iterative alignment and consistency-based alignment.

In *progressive alignment*, the closest sequences are aligned first, then more distant sequences are added. At its most basic, the progressive method follows the following process outlined by Feng and Doolittle [14]:

- Create Pairwise Alignments: Each pair is compared with every other pair (using, for example, the Needleman-Wunsch algorithm), and the results, such as a distance matrix, are stored.
- Identify Closest Related Pair: The results of the pairwise alignment are ordered, and the closest pair is identified.
- Insert Neutral Elements: The closest pair (e.g. A and B) is aligned, and gaps are inserted as necessary. Then the next closest relative (e.g., C) is identified, and it is aligned with the results of the previous alignment<sup>1</sup>. This process is repeated until all sequences are brought into the alignment. Note that if, at any repetition, a gap is introduced when forming an alignment, it cannot be removed later.
- Score Alignment: The final alignment is scored.
- Construct Tree: Branching order is determined, and branch lengths calculated. Minor adjustments are made at this point to ensure that negative branch lengths do not occur. A guide tree (or dendrogram) is created from the alignment, to identify the distances of sequences from one another.

A significant problem in progressive alignment is that, if an incorrect alignment is made early in the process, that error does not get fixed (recall from the description above that gaps, once introduced, are never removed).

In *iterative alignment*, a progressive alignment is created, and then the program iterates, making improvements until either the maximum number of iterations is reached, or until no further improvements can be made.

Thompson, et. al. [52] note that iterative programs can produce improved alignments, but with the trade-off of longer computation time.

In *consistency-based alignment*, the idea is that the pairwise alignments are consistent across all sequences in the set, i.e., if Sequence A, position 10

---

<sup>1</sup>While for this example, we use a scenario in which A, B, and C are individual sequences, it is also possible that these are, in fact, alignments of other sequences. For example, A is an alignment of sequences L, M, and N, B is an alignment of P, Q, and R, and C is an alignment of X, Y, and Z.

aligns with Sequence B, position 12, and this position aligns with Sequence C position 20, then in the final alignment, Sequence A, position 10 will align with Sequence C, position 20. In this approach, both global and local alignments are created, and each pairwise score is weighted and added. The scores are added, and the weights recalculated if necessary, to create a single library of scores. The pairs in this library are then used when comparing a given pair in all the sequences to be aligned.

### 6.2.1 Programs for Multiple Sequence Alignment

Note that this was not an exhaustive examination of every MSA program in use. However, every effort was made to identify the widely used programs that would be applicable to the problem at hand.

#### CLUSTALW

CLUSTALW (v1.83) [50] builds its version of a progressive alignment by doing a pairwise comparison of all the sequences in the set and then building a guide tree using neighbor joining. Each sequence is then aligned using the branching order in the guide tree.

The CLUSTALX program provides a graphical (XWindows) user interface for CLUSTALW.

#### DIALIGN

DIALIGN (v2.2.2) [31] does a combination of global and local alignment, aligning portions of sequences that are related, and leaving what it considers to be unrelated portions of the sequences unaligned. It proceeds by identifying sequence segments that show some degree of similarity.

DIALIGN is also interesting in that it allows those with expert knowledge of the sequences to identify alignment anchors, or sequences that must be aligned to each other, or to identify sequences that should be excluded from the alignment.

DIALIGN compiled under cygwin, but was not used as part of this study. In testing for this project, DIALIGN appears to work reliably, but was quite slow for large sequence sets; for example, an attempt to sequence the set of 500 sequences of approximately 500 nucleotides in length was stopped after

31 hours of processing without result. Edgar [10] corroborates this result, pointing out that DIALIGN is primarily useful for fewer than 100 sequences.

## **MAFFT**

The MAFFT (v6.240) program [23] uses homologous segments (i.e., segments that are structurally similar) within a sequence set to speed up the alignment. Finding the correlation between two sequences is done using a Fast Fourier Transform (FFT). In addition, sequencing is considerably speeded by reducing the complexity of the scoring algorithm.

MAFFT compiled and ran under cygwin. While it has a variety of approaches to alignment, this study examined only those that appeared capable of handling 1000+ sequences: the progressive model (FFT-NS-2), purported to replicate CLUSTALW speed and accuracy, and the iterative model (FFT-NS-i). We note that MAFFT includes a number of approaches to alignment that focus on the alignment of smaller sequences (i.e., less than 200 sequences in the set).

## **MUSCLE**

Muscle (v3.6) [8] first builds a progressive alignment. It computes the similarity of each pair of sequences, and a distance matrix, from which a guide tree is constructed (using either neighbor joining or UPGMA). The guide tree is used to construct a preliminary alignment.

In a second phase, the algorithm attempts to improve the tree. This step may include iteration, and is considered complete when an iteration does not change any nodes (i.e., does not find any improvement). In this phase, the pairwise similarities are recomputed, and a new tree is created and compared to the original tree.

In the third phase, iterative refinement is performed by removing an edge to partition the tree, and a profile-profile realignment is performed. This phase attempts to maximize the “Sum of Pairs” score (see Section 6.2.2).

MUSCLE ran reliably under Windows XP on large sets of sequences, and so was used as part of this study.

## **ProbCons**

ProbCons (v1.12) [4] uses a method called probabilistic consistency, for use in scoring and comparing MSAs, to improve on the consistency-based alignment

approach.

ProbCons compiled easily under cygwin, and produced several alignments from the sample data. However, performance on larger sets appears problematic. For example, the set of 1000 sequences of approximately 1000 nucleotides in length ran for approximately 28 hours and, at its peak consumed over 60% of RAM, before doing a core dump. Due to the erratic performance, it was not selected for this project. Edgar [10] points out that ProbCons is highly accurate, but is primarily useful for fewer than 100 sequences.

### T-Coffee

T-Coffee (v5.05) [33] is the best known consistency-based alignment program. It uses CLUSTALW to create the library of global alignments, and LALIGN to create the local alignments. T-Coffee is considered one of the more accurate programs, but like ProbCons and DIALIGN, it does not scale well for more than 100 sequences.

T-Coffee was originally considered as one of the programs to test. However, while it compiled under cygwin, it ran extremely slowly, and hung on more than one occasion. Therefore, it was not used.

## 6.2.2 Comparing Multiple Sequence Alignments

Commonly used metrics for comparing the quality of multiple sequence alignments are the *Sum-of-Pairs* score, the *Modeler* score, the *Shift* score, and the *Total Column* score.

The Sum of Pairs (SP) score increases with the increase in the number of sequences correctly aligned. This is also referred to as the *Developer* (or  $F_D$ ) score in Sauder, *et. al.* [45], where it is defined as “the ratio of the number of residues correctly aligned in the sequence alignment, divided by the total number of aligned residues in the structure alignment.” Here, the *structure alignment* is analogous to the known, or reference, sequence, while the *sequence alignment* is the test sequence being compared. It is known as the Developer score because it is the score of interest to the algorithm developer: how well does the alignment produced match the correct alignment?

A common heuristic in MSA programs is to attempt to maximize the SP score [7].

The *Modeler* (or  $F_M$ ) score is “the number of amino acids correctly aligned in the sequence alignment divided by the total number of aligned residues in



the sequence alignment” [45]. It is considered a reflection of the quality of the generated model (when the accuracy of the model cannot be determined).

In the above, given nine pairs in the structure alignment,  $F_D = \frac{5}{9} = .555$ , and  $F_M = \frac{5}{6} = .833$ .

The shift score is described in Cline *et. al.* [3] as reflecting “many types of alignment error including misalignment, aligning too much, and aligning too little,” and as superior to other metrics used for the same purpose. The scale is from 1.0 (reflecting an accurate alignment) to  $-\epsilon$ , a scoring parameter typically set at 0.2<sup>2</sup>. The algorithm itself is fairly complex, and interested readers are referred to the paper.

The Total Column score (TC) is a reflection of the program’s ability to correctly align all sequences. For each column in the alignment, the column is either scored as ‘1’ if the column is matched in the structure alignment, and ‘0’ if not. These scores are then summed, and divided by the total number of columns [52].

### 6.2.3 Literature Search

The topic of creating MSAs is quite active. New programs have been introduced in the past few years, as have new methods for comparing the quality of alignments produced. In this section, we examine some of the more recent studies that compare various algorithms. The studies tended to reinforce one another, and there were no significant disparities between studies. The bulk of the evidence appears to indicate that MAFFT is considered the superior program, although the specific algorithm within MAFFT must be chosen appropriately. When working with smaller sequence sets, ProbCons was the most accurate.

Thompson *et. al.* [52], one of the authors of the CLUSTAL family of programs, evaluated 10 programs (PRRP, CLUSTALX, SAGA, MULTALIGN, PILEUP8, SBPIMA, DIALIGN, MLPIMA, MULTAL, HMMT) in 1999. Using the BAliBASE benchmark [51], they found CLUSTALX (progressive) and DIALIGN, PRRP, and SAGA (iterative) to be the best for the specific alignment tasks tested. This study used the sum-of-pairs (SP) and total column (TC) scores as the basis of comparison.

Katoh *et. al.* [23], as part of the paper describing MAFFT, compares it to

---

<sup>2</sup>This has nothing to do with and should not be confused with *machine epsilon*. Here, epsilon is simply a variable, and the bottom of the shift score scale is its inverse.

CLUSTALW and T-Coffee, using ROSE-generated data and the BALiBASE benchmark set. The study finds MAFFT’s NW-NS-2 algorithm produces an alignment of comparable accuracy to CLUSTALW, and MAFFT’s NW-NS-i algorithm produces an alignment of comparable accuracy to T-Coffee, and both results are produced in less time.

Do *et. al.* [4], as part of the paper introducing ProbCons, compared it to Align-m, DIALIGN, CLUSTALW, MAFFT, T-Coffee, and MUSCLE, using the BALiBASE benchmark set. ProbCons was the most accurate program, although it was not the fastest.

Ahola *et. al.* [1], as part of a paper introducing the alignment quality (AQ) score for the comparison of alignments, compared CLUSTALW, DIALIGN, MAFFT, MUSCLE, ProbCons, and T-Coffee, using the BALiBASE benchmark set. MAFFT (L-INS-i) was considered superior to the other methods.

Edgar [10] compared CLUSTALW (cited as “the most widely used MSA program”) with what the author regarded as the best comparable programs: MAFFT, MUSCLE, T-Coffee, and ProbCons. ProbCons and T-Coffee are cited as having a practical maximum of approximately 100 sequences, due to CPU and memory requirements. Testing, using BALiBASE, suggests that for sequence sets of the size being aligned in this project, MUSCLE or MAFFT are the most appropriate.

The Edgar study was also useful for its summarization of specific alignment tasks, and the appropriate alignment programs for each (shown verbatim in Table 6.1 and Table 6.2). The tables also point up the fact that the MSA program used should depend upon the nature of the sequence set being analyzed; sequence analysis should not be a matter of having only one MSA program available, and using it exclusively.

Nuin *et. al.* [34], compared CLUSTALW (v1.8), DIALIGN (v2.2), T-Coffee (v3.27), POA (v2.0), MUSCLE (v3.6), MAFFT (v5.732), ProbCons (v1.1), DIALIGN-T (v0.2.1), and Kalign (v1.04). Comparisons were done using both BALiBASE and Simprot-generated sequences. ProbCons and MAFFT (L-INS-i algorithm) were considered the best, and were roughly equivalent in accuracy; POA, both versions of DIALIGN, and CLUSTALW

Program	Advantages	Cautions
CLUSTALW	Uses less memory than other programs	Less accurate or scalable than modern programs
DIALIGN	Attempts to distinguish between alignable and non-alignable regions	Less accurate than CLUSTALW on global benchmarks
MAFFT, MUSCLE	Faster and more accurate than CLUSTALW; good trade-off of accuracy and computational cost. Options to run even faster, with lower average accuracy, for high-throughput applications.	For very large data sets (say, more than 1000 sequences) select time- and memory-saving options
PROBCONS	Highest accuracy score on several benchmarks	Computation time and memory usage is a limiting factor for large alignment problems (>100 sequences)
ProDA	Does not assume global alignability; allows repeated, shuffled and absent domains.	High computational cost and less accurate than CLUSTALW on global benchmarks
T-COFFEE	High accuracy and the ability to incorporate heterogeneous types of information	Computation time and memory usage is a limiting factor for large alignment problems (>100 sequences)

Table 6.1: Summary of MSA Programs (from Edgar, 2006 [10])

Input Data	Recommendations
2-100 sequences of typical protein length (maximum around 10,000 residues) that are approximately globally alignable	Use PROBCONS, T-COFFEE, and MAFFT or MUSCLE, compare the results using ALTAVIST. Regions of agreement are more likely to be correct. For sequences with low percent identity, PROBCONS is generally the most accurate, but incorporating structure information (where available) via 3DCoffee (a variant of T-COFFEE) can be extremely helpful.
100-500 sequences that are approximately globally alignable	Use MUSCLE or one of the MAFFT scripts with default options. Comparison using ALTAVIST is possible, but the results are hard to interpret with larger numbers of sequences unless they are highly similar.
>500 sequences that are approximately globally alignable	Use MUSCLE with a faster option (we recommend maxiters-2) or one of the faster MAFFT scripts
Large numbers of alignments, high-throughput pipeline.	Use MUSCLE with faster options (e.g. maxiters-1 or maxiters-2) or one of the faster MAFFT scripts
2-100 sequences with conserved core regions surrounded by variable regions that are not alignable	Use DIALIGN
2-100 sequences with one or more common domains that may be shuffled, repeated or absent.	Use ProDA
A small number of unusually long sequences (say, >20,000 residues)	Use CLUSTALW. Other programs may run out of memory, causing an abort (e.g. a segmentation fault).

Table 6.2: Typical Alignment Tasks (from Edgar, 2006 [10])

were considered the worst. On the basis of both speed and accuracy, MAFFT was considered the best overall.

Perez-Losada *et. al.* [39] made the point that alignment programs embody certain approaches, and that these approaches may work well on some sequences, but not on others. The authors recommend always using a variety of MSA programs. Their results indicated that, for accuracy, the best programs were (ordered) MAFFT, MUSCLE, T-Coffee, and CLUSTALW; from the speed perspective, the best programs were (ordered) MUSCLE, MAFFT, CLUSTALW, and T-Coffee.

## 6.3 Methods

### 6.3.1 Computer Environment

All sequencing and tree generation activities were performed on a Lenovo T60 laptop, with dual T2400 1.83GHz processors, and 2GB of RAM. The operating system was Windows XP Professional (Service Pack 2), and except where noted, all processing took place under Windows XP.

It should be noted that execution times for various activities should be used as a guideline only. The computer was not dedicated to the task, and other CPU-intensive tasks of varying duration may have executed during the course of, for example, a single PHYML run. In short, the situation was analogous to what would be encountered on any other multi-user machine.

### 6.3.2 Generation of data sets

The data sets used were the same as those used in the sub-project on evaluating phylogenetic trees for this clinic, generated using the ROSE program [46]. ROSE uses a probabilistic model, easily adjustable through configuration files, to generate DNA sequences using one of several possible evolutionary models. In the process of generating sequence data, the program creates the true phylogenetic tree, and the correct sequence alignment. For this project, the default settings of ROSE were used wherever possible. The Jukes-Cantor evolutionary model was used.

The author learned too late of a variety of benchmark DNA sets (Edgar [8] mentions the BALiBASE, SABmark, SMART, and PREFAB sets), one of

which may have been preferable to generating new sequences (although most appear to focus on smaller sequence sets).

The author also learned too late of Simprot [37], another software package that creates simulated sequences.

### 6.3.3 Aligning and Scoring

The programs tested against CLUSTALW were MUSCLE and MAFFT, on the grounds (no T-Coffee pun intended) that these were the alignment programs that could be made to run under either Windows or cygwin (a Unix environment running under Windows), and that were considered appropriate for performing the MSA on the LANL set of sequences. A number of programs reputed to be excellent in aligning shorter sets were not evaluated.

For this study, Edgar's *qscore* program [9] was used.

### 6.3.4 Generating Phylogenetic Trees

To produce useful results for inclusion in the Math Clinic final report, Maximum Likelihood trees were created for those sequence sets with (approximate) length of 1000 nucleotides, on the grounds that these sequences would most closely resemble the LANL data set under consideration.

PHYML [16] was used to generate the maximum likelihood trees, after problems with hanging were encountered using the DNAML program in PHYLIP [11].

The accuracy of the generated trees was scored using the TREEDIST program in PHYLIP, which calculates symmetric difference (also known as the Robinson-Foulds (RF) distance [41]) and Branch Score Distance (BSD) [27], two of the most commonly used metrics for comparing phylogenetic trees. Log Likelihood scores were produced by PHYML, as part of the tree generation process.

## 6.4 Results

Using the data sets generated by ROSE, MSAs were run using CLUSTALW, MAFFT (both the default NS-2 algorithm, as well as the NS-i algorithm), and MUSCLE. Generation times are shown in Figure 6.1.

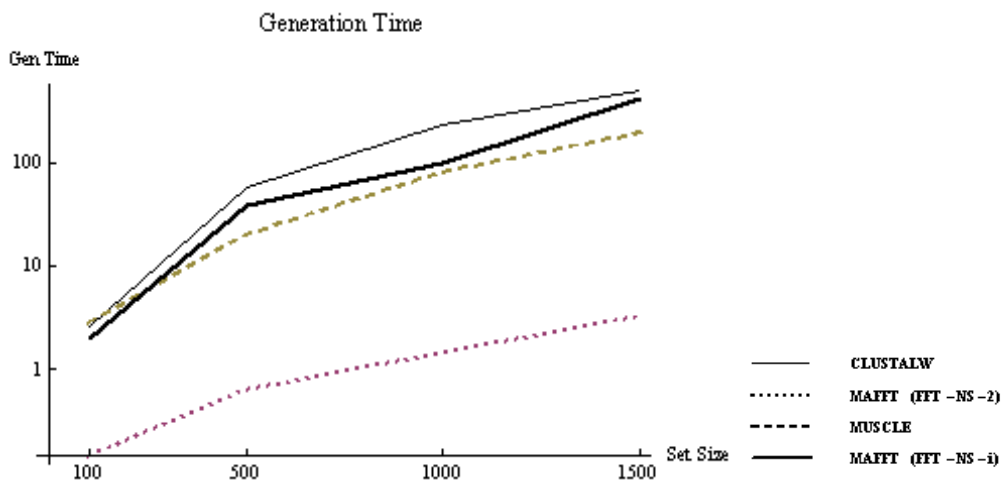


Figure 6.1: MSA Generation Times (minutes)

Using the qscore program, the MSAs produced were compared with the “correct” MSAs generated by ROSE. The scores are shown in Table 6.3. Shift scores were graphed, and are shown in Figure 6.2 through Figure 6.5.

We then used the sets with 100, 500, 1000, and 1000 sequences (all 1000 nucleotides in length) and generated phylogenetic trees, using PHYML. The Robinson Foulds (RF) distances are shown in Figure 6.6, the Branch Score Distances (BSD) are shown in Figure 6.7, and the log likelihood distances are shown in Figure 6.8.

## 6.5 Discussion

It appears clear, from the shift scores collected during the evaluation, that MAFFT’s NS-i algorithm is capable of generating an alignment at least as good as CLUSTALW, for data sets of up to 1500 sequences. It is able to generate these alignments in significantly less time. These findings lead us to believe that MAFFT is a viable alternative to CLUSTALW for the problem of aligning a set of over a thousand sequences, each over 1000 nucleotides in length.

Set	GLUSTALW				MAFPPT (FPPT-NS-2)				Muscle (default)				MAFPPT (FPPT-NS-1)			
	SP	Modeler	Shift	TC	SP	Modeler	Shift	TC	SP	Modeler	Shift	TC	SP	Modeler	Shift	TC
NS1000L50	0.141	0.144	0.222	0.0893	0.645	0.668	0.691	0.107	0.726	0.723	0.752	0	0.811	0.819	0.843	0.125
NS1000L100	0.109	0.109	0.172	0.15	0.179	0.188	0.21	0.121	0.0803	0.0804	0.0728	0.0429	0.344	0.338	0.397	0.236
NS1000L500	0.0645	0.0651	0.0429	0.233	0.0376	0.0394	-0.0267	0.156	0.0232	0.0245	-0.0585	0.0712	0.0604	0.0597	0.0302	0.169
NS1000L1000	0.0629	0.063	0.0289	0.192	0.0398	0.0414	-0.0193	0.125	0.014	0.015	-0.0928	0.0223	0.0646	0.0636	0.0538	0.116
NS500L50	0.154	0.158	0.268	0.488	0.255	0.291	0.331	0.209	0.606	0.603	0.62	0.109	0.432	0.447	0.434	0.256
NS500L100	0.113	0.114	0.2	0.484	0.0965	0.103	0.132	0.284	0.0605	0.0593	0.0977	0.124	0.45	0.445	0.492	0.298
NS500L500	0.0674	0.0688	0.0651	0.546	0.0427	0.046	-0.000897	0.4	0.0288	0.0285	-0.0304	0.0777	0.137	0.134	0.18	0.376
NS500L1000	0.0533	0.0542	0.0269	0.524	0.0215	0.023	-0.0563	0.345	0.0138	0.0138	-0.084	0.0308	0.111	0.109	0.118	0.345
NS1000L50	0.128	0.133	0.245	0.509	0.118	0.155	0.183	0.292	0.177	0.176	0.284	0.117	0.616	0.643	0.64	0.48
NS1000L100	0.116	0.12	0.199	0.672	0.136	0.165	0.176	0.436	0.0656	0.0647	0.0639	0.147	0.558	0.549	0.564	0.554
NS1000L500	0.0586	0.0611	0.0517	0.662	0.0311	0.0351	-0.0252	0.419	0.0313	0.0307	-0.00186	0.104	0.0976	0.101	0.0605	0.393
NS1000L1000	0.0471	0.0486	0.00286	0.668	0.0176	0.0193	-0.0735	0.434	0.0248	0.0242	-0.0356	0.0579	0.041	0.0402	0.0181	0.41
NS1500L50	0.151	0.155	0.275	0.591	0.105	0.137	0.177	0.34	0.448	0.447	0.523	0.217	0.214	0.249	0.271	0.36
NS1500L100	0.0846	0.0873	0.144	0.659	0.0756	0.0933	0.0922	0.394	0.457	0.45	0.489	0.329	0.322	0.336	0.362	0.524
NS1500L500	0.0431	0.0451	0.0206	0.716	0.0185	0.0211	-0.0605	0.425	0.0665	0.0653	0.0399	0.107	0.0899	0.0903	0.102	0.403
NS1500L1000	0.0483	0.0505	0.0332	0.729	0.0132	0.0153	-0.0875	0.41	0.0196	0.0194	-0.0542	0.0711	0.0537	0.0555	-0.00298	0.404

Table 6.3: Scores for Generated MSAs



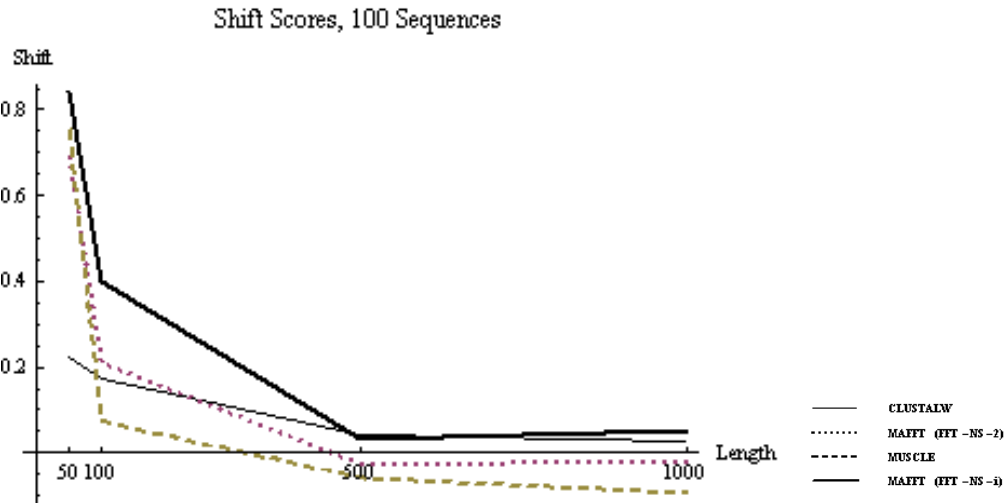


Figure 6.2: Shift scores, 100 sequences

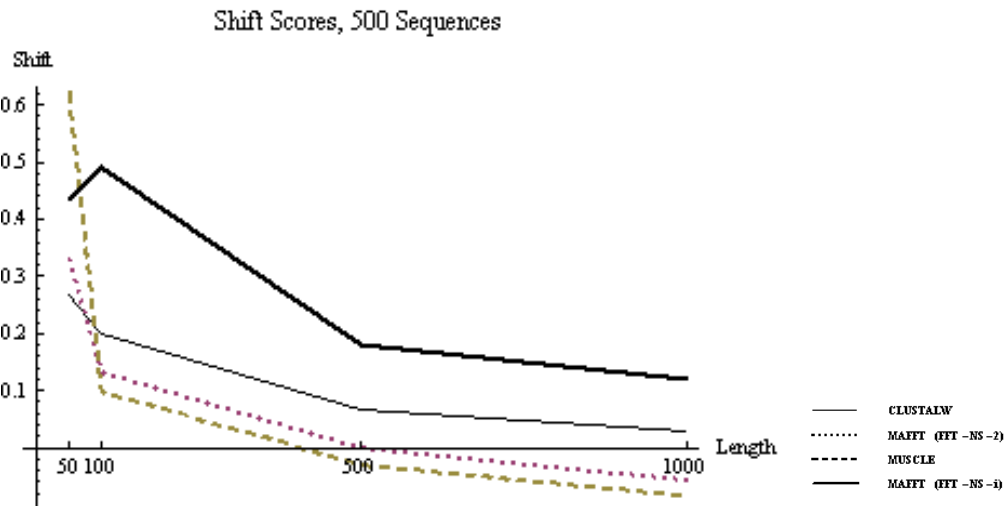


Figure 6.3: Shift scores, 500 sequences

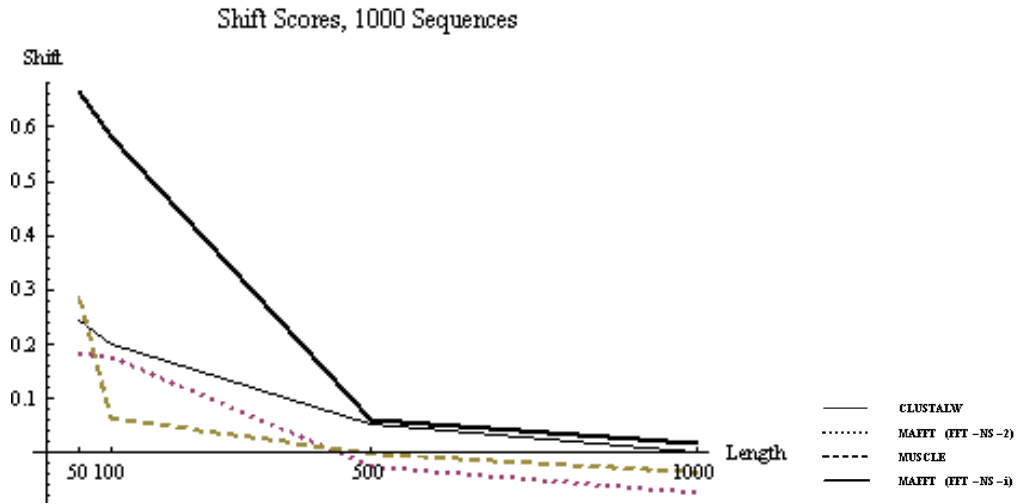


Figure 6.4: Shift scores, 1000 sequences

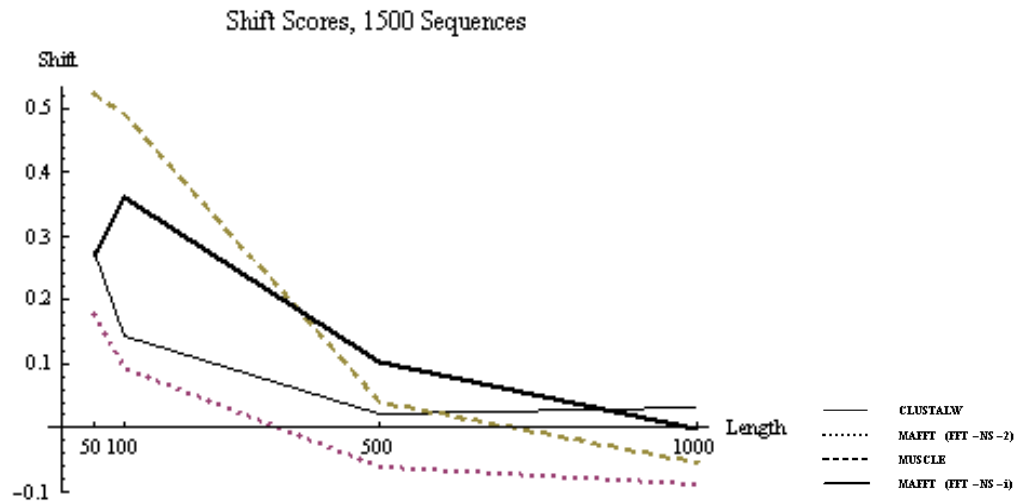


Figure 6.5: Shift scores, 1500 sequences

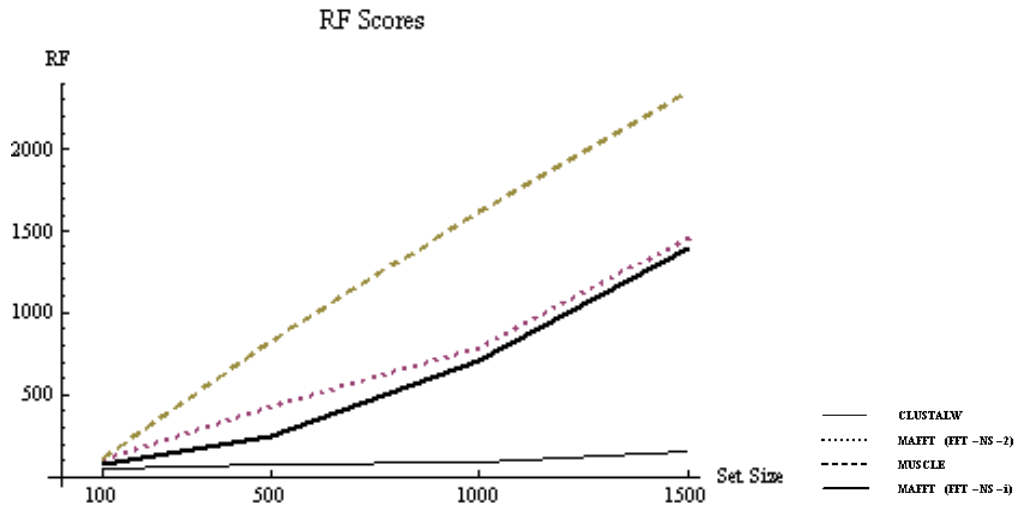


Figure 6.6: RF Distance: Set Sizes 100, 500, 1000, 1500; Sequence Length 1000



Figure 6.7: Branch Score Distance: Set Sizes 100, 500, 1000, 1500; Sequence Length 1000

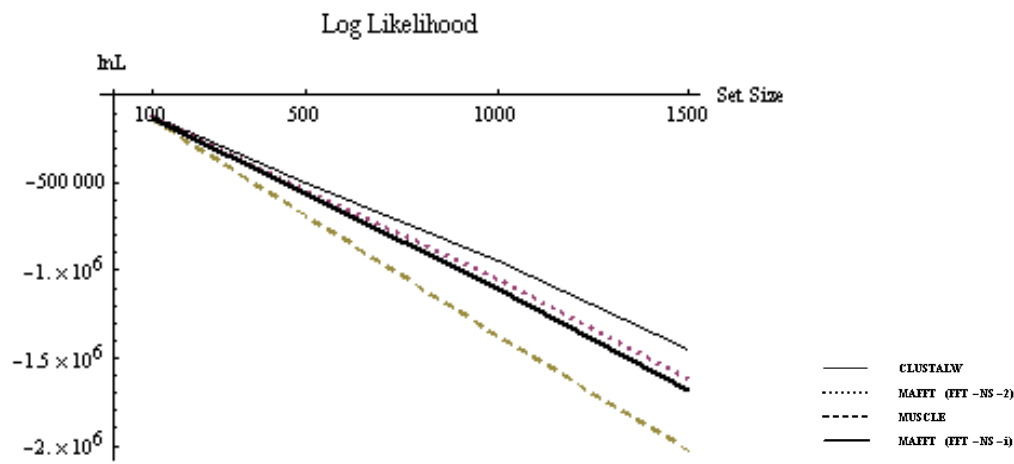


Figure 6.8: Log Likelihood Scores: Set Sizes 100, 500, 1000, 1500; Sequence Length 1000

These findings for MAFFT are supported by the available literature.

The MAFFT FFT-NS-2 (progressive) method was claimed to be much faster than CLUSTALW, while producing equally accurate results. While generation times for the NS-2 algorithm were extremely impressive, our experiments did not validate the level of accuracy claimed: MSA results were almost uniformly inferior (with regard to the shift score) and the resulting phylogenetic trees appear considerably worse from a topological perspective (as represented by the RF score).

Experimental results in using MUSCLE were even less encouraging. Use of MUSCLE with its default settings led to alignments that were occasionally superior to CLUSTALW and MAFFT, especially at shorter sequence lengths, but generally inferior to both (using shift score as the basis of comparison). Use of the MUSCLE *maxhours* parameter, which supposedly attempts to generate the best alignment possible within the time allotted, led to the creation of a virtually identical set of alignments.<sup>3</sup> While it may be possible to generate better alignments with MUSCLE using different command-line settings, this would require significant additional experimentation.

These findings for MUSCLE are not supported by the available litera-

<sup>3</sup>The sole differences were in the S1500L50 and S1500L100 sets, where there was a very slight decrease in quality when using the *maxhours* option. For this reason, these data points were not included in this report.

ture. In [10], MAFFT, MUSCLE, ProbCons, and T-Coffee are rated as most accurate, and “new methods outperform the CLUSTALW tool in terms of average accuracy.”

This study became more problematic when investigating the hypothesis that a better alignment should produce a better phylogenetic tree. Using PHYML to produce maximum likelihood trees, we did not observe that alignments with better shift scores produced “better” phylogenetic trees; the trees generated using the CLUSTALW alignments were better in terms of RF and BSD score, and well within an order of magnitude for the log likelihood score.<sup>4</sup>

In the hope that the issue might have been the use of PHYML, Neighbor Joining trees were produced, using BIONJ [15], and MatLab’s Bioinformatics Toolkit, but both programs produced identical results for all alignments of a given set size, and so were not used in this study.

The specific instances in which findings from the literature could not be replicated are troubling. The most straightforward explanation is that the benchmark data sets used in the various comparisons in the literature are sufficiently different from the ROSE-generated data sets used here that the results from this experiment are not comparable.

A less likely explanation is that “accuracy” in a phylogenetic tree is not positively correlated with any of the metrics (RF, BSD, or  $\ln(L)$ ) used to compare the generated trees with the known tree. However, since these metrics have been widely used in the domain for one or more decades, this explanation is rather far-fetched.

### 6.5.1 Future Work

This paper found results in software performance that were contrary to the literature, and the finding that the most accurate MSA did not produce the most accurate phylogenetic tree is quite surprising. This latter result, especially, should be followed up, with the repetition and expansion of some of the experiments from this study.

The hypothesis of the follow-on study would be that the most accurate MSA produces the most accurate phylogenetic tree, and would use the following experiments to test the hypothesis:

---

<sup>4</sup>However, it is worthwhile to mention that we cannot definitively state that a difference of  $n$  points in a given scale is a “significant” difference.

- Using the BALiBASE benchmark data set, create alignments for one or more sets of 100 sequences, using CLUSTALW, MUSCLE, MAFFT, T-Coffee, and ProbCons.
- Using *qscore*, score the obtained alignments. Since the literature cites a number of studies using these alignment programs with BALiBASE, we would expect to see the results for relative accuracy to correlate to those in the literature.
- Using PHYLIP, create an assortment of phylogenetic trees (neighbor-joining, UPGMA, maximum likelihood, parsimony).
- Using PHYLIP, score the trees using RF and branch score distances.
- Assuming that the hypothesis can be verified for smaller sequence sets, proceed to investigate whether the results hold for larger sets (250, 500, 750, 1000, and 1250). The larger sets would be obtained from known benchmark data sets if at all possible, with the backup plan being to generate the data sets using ROSE or Simprot.

This study has also suggested a variety of other follow-on efforts, any one of which might be considered a contribution to the literature:

- An examination of the various scoring metrics for MSA accuracy;
- An examination of the available MSA programs, studying accuracy, speed, and the range of utility (i.e., how many sequences, limits on sequence length);
- An in-depth examination of MAFFT's various algorithms, their speed, accuracy, and applicability to specific alignment problems;
- An examination of the effects of accuracy of MSA on downstream processes (e.g., phylogenetic tree generation, using a variety of algorithms); and
- Scoring metrics for phylogenetic trees.

## 6.6 Conclusions

From the data presented above on using CLUSTALW, MAFFT, and MUSCLE *on their default settings*, we conclude that CLUSTALW is the superior choice for creating an MSA.

For users willing to learn and experiment with command line parameters, MAFFT, using the NS-i option, is able to produce alignments that are superior to CLUSTALW alignments, in less time.

We are unable to conclude at this point whether a “better” multiple sequence alignment produces a “better” output in a downstream process using the MSA, such as phylogenetic tree generation.

For the project at hand, we would recommend using the MAFFT program’s NS-i algorithm to build the MSA for the LANL data set.





# Chapter 7

## Conclusions

The primary goal of this project was to develop and train a mathematical model of nucleotide mutation rates for the gene encoding for Neuraminidase in the H5N1 (Bird Flu) virus. The model we implemented uses a general time reversible model, a widely used model in the field. This model has 10 parameters, but constraints on the model result in 8 degrees of freedom. The model is described in detail in Chapter 2.

Parameters for the model were trained using Neuraminidase DNA sequences for the Los Alamos National Laboratory Influenza Sequence Database [30]. This was accomplished by first performing a multiple sequence alignment (using CLUSTALW) and then inferring a phylogenetic tree for the sequences, which approximates the evolutionary relationships between the sequences. With the phylogenetic tree thus determined, the parameters of the model were calculated using a maximum likelihood method.

The approach is sensitive to the evolutionary inferences made via the multiple sequence alignment and the phylogenetic tree construction methodologies. An important component of our work, therefore, was to investigate various methodologies available in the literature. Two different studies on phylogenetic trees were considered. The first study, described in Chapter 3 compares a number of algorithms using simulated data to determine which algorithmic approach most accurately reproduces the “true” phylogenetic tree. Preliminary results indicate that GARLI [58] using a ML start produces the most accurate trees, as measured by three different methods for measuring tree accuracy. But a faster algorithm, PHYML [16], is nearly as accurate, but runs in considerably less time.

However, there was a limitation in this study that requires further in-

vestigation. In particular, the simulated data was produced using a Jukes Cantor model for mutation rates. This has the effect of making the Hamming distances between sequences be a very good proxy for the phylogenetic distances. As such, relatively simple algorithms for tree reconstruction are able to do a fairly good job. Thus, the advantages in accuracy of the more sophisticated algorithms, such as GARLI, may be difficult to realize. Repeating the experiments using a more sophisticated model (such as the general time reversal model) for generating the data may provide clearly results.

The second study, described in Chapter 4 assessed the importance of tree topology on the accuracy of the model. Most of the computational time required by the more accurate phylogenetic algorithms (such as GARLI) is spent searching over various tree topologies. This study focused on whether getting the tree topology correct is worth the extensive computational time. To address this question, different algorithms were used to infer phylogenetic trees from simulated data. Differences in the resulting tree topologies were then measured, and compared to differences in pairwise distances between leaf nodes. The study concluded that the accuracy of the pairwise distances is very sensitive to the tree topology, so there appears to be significant advantage in getting the tree topology correct.

One limitation of the study was that “true” phylogenetic distances were used for the all of the tree building algorithms. As a result, most of the phylogenetic algorithms tested were able to recover the exact tree topology very accurately. An improved study, which is left for future work, is to first generate random sequences from the trees, and using the Hamming distances between these sequences as the distances supplied to the phylogenetic algorithms.

One conceptual issue that had to be addressed in this study was how to measure differences between pairwise distances in trees. Toward that end, an innovative new metric was developed, which was based solely on precedence relations between pairwise-distances.

Another study, described in Chapter 6 evaluates recent algorithms for generating multiple sequence alignments. This study concluded that the MAFFT algorithm using the NS-option produces the most accurate multiple sequence alignments, and it is considerably faster than the industry standard CLUSTALW algorithm. However, using the default options in MAFFT does not produce as accurate of results as CLUSTALW, so the use of MAFFT is only recommended for sophisticated users.

An interesting question raised in this study is whether better multiple se-

quence alignments will ultimately produce more accurate phylogenetic trees or more accurate mutation models. Interestingly, in this study, it was observed that better scoring multiple sequence alignments did not appear to produce more accurate phylogenetic trees. If this observation is confirmed through additional experimentation, it raises the question of whether current measurements of the quality of a multiple sequence alignment really make sense.



# Bibliography

- [1] V. Ahola, T. Aittokallio, M. Vihinen, and E. Uusipaikka. A statistical score for assessing the quality of multiple sequence alignments. *BMC Bioinformatics*, 7:484, 2006.

Compares MAFFT, ProbCons, T-Coffee, and MUSCLE, in the context of defining a new score for comparing MSA quality.

- [2] R. Cartwright. Dna assembly with gaps (dawg): Simulating sequence evolution. *Bioinformatics*, 21:iii31–iii38, 2005.

A program that assigns DNA sequences to a tree with branch lengths. See also the web site <http://scit.us/projects/dawg/>.

- [3] M. Cline, R. Hughey, and K. Karplus. Predicting reliable regions in protein sequence alignments. *Bioinformatics*, 18(2):306–314, 2002.

Reference for the Shift score

- [4] C.B. Do, M.S.P. Mahabhashyam, M. Brudno, and S. Batzoglou. Probcons: Probabilistic consistency-based multiple sequence alignment. *Genome Research*, 15:330–340, 2005.

Primary PROBCONS reference

- [5] Alexei J. Drummond, Geoff K. Nicholls, Allen G. Rodrigo, and Wiremu Solomon. Estimating mutation parameters, population history and genealogy simultaneously from temporally spaced sequence data. *Genetics*, 161(3):1307–1320, 2002.

- [6] R.A. Dwyer. *Genomic Perl*. Cambridge University Press, Cambridge, UK, 2003.

Good, hands-on introduction to programming in bioinformatics

- [7] R.C. Edgar. Muscle: a multiple sequence alignment method with reduced time and space complexity. *BMC Bioinformatics*, 5:113, 2004.

Describes the MUSCLE algorithm.

- [8] R.C. Edgar. Muscle: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, 32:1792–1797, 2004.

Primary article on MUSCLE.

- [9] R.C. Edgar. qscore v1.1, 2007.

No reference found for program, and no user manual (other than what you see by typing qscore -help). No dates or other information in the source code. Downloaded 10/31/2007.

- [10] R.C. Edgar and S. Batzoglou. Multiple sequence alignment. *Current Opinions in Structural Biology*, 16:368–373, 2006.

Excellent summary table of advantages and disadvantages of current programs, along with a table summarizing what to do for specific alignment tasks. Very good annotated bibliography.

- [11] J. Felsenstein. Phylip - phylogeny inference package (version 3.2). *Cladistics*, 5:164–166, 1989.

There is no good citation for PHYLIP. This is the one that the PHYLIP web site says to use, but Felsenstein admits that it's out of date, and that the 'real' citation is simply the web site (<http://evolution.genetics.washington.edu/phylip.html>).

- [12] J. Felsenstein. Phylip - phylogeny inference package (version 3.2). *Cladistics*, 5:164–166, 1989.

The collection of programs known as PHYLIP. See also the web site <http://evolution.genetics.washington.edu/phylip.html>.

- [13] Joseph Felsenstein. Evolutionary trees from dna sequences: a maximum likelihood approach. *Journal of molecular evolution*, 17:368–376, 1981.
- [14] D.F. Feng and R.F. Doolittle. Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *Journal of Molecular Evolution*, 25:351–360, 1987.

Seminal article on progressive sequence alignment.

- [15] O. Gascuel. Bionj: an improved version of the nj algorithm based on a simple model of sequence data. *Molecular Biology and Evolution*, 14:685–695, 1997.

Main reference for BIONJ.

- [16] S. Guindon and O. Gascuel. A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood. *Systematic Biology*, 52(5):696–704, 2003.

Primary PHYML paper

- [17] M. Hasegawa, H. Kishino, and T. Yano. Dating of the human-ape splitting by a molecular clock of mitochondrial dna. *Journal of molecular evolution*, 22:160–175, 1985.
- [18] Red Hat. Cygwin documentation. Web site, Red Hat, Inc., 2001. <http://cygwin.com/>.

Cygwin is an online enterprise. Documentation available for download at <http://cygwin.com/docs.html>.

- [19] V. Hollich, L. Milchert, L. Arvstad, and E.L.L. Sonnhammer. Assessment of protein distance measures and tree-building methods for phylogenetic tree reconstruction. *Molecular Biology and Evolution*, 22(11):2257–2264, 2005.

Compares NJ and variants (NJ, BIONJ, FastME, Weighbor)

- [20] Jack Horner. Math clinic research project overview. Presentation to the Math Clinic class, August 2007.

- [21] Bioinformatics Group in the Department of Computer Science. Phylonet: Phylogenetic networks toolkit. User documentation, Rice University, 2007. <http://bioinfo.cs.rice.edu/>.

Unable to locate any citations for this work, other than its (very good) user documentation, which is included in the PhyloNet download.

- [22] T. H. Jukes and C. R. Cantor. Evolution of protein molecules. In H. N. Munro, editor, *Mammalian Protein Metabolism*, pages 21–123. Academic Press, New York, 1969.
- [23] K. Katoh, K. Misawa, K. Kuma, and T. Miyata. Mafft: a novel method for rapid multiple sequence alignment based on fast fourier transform. *Nucleic Acids Research*, 30:3059–3066, 2002.

Primary MAFFT reference

- [24] M. Kimura. A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. *Journal of molecular evolution*, 16:111–120, 1980.
- [25] M. Kimura. Estimation of evolutionary distances between homologous nucleotide sequences. *Proceedings of the National Academy of Sciences of the United States of America*, 78:454–458, 1981.
- [26] M. Kuhner and J. Felsenstein. A simulation comparison of phylogeny algorithms under equal and unequal evolutionary rates. *Molecular Biology and Evolution*, 11:459–468, 1994.

This paper introduces the method of branch score distance between the topologies of two phylogenetic trees.

- [27] M.K. Kuhner and J. Felsenstein. A simulation comparison of phylogeny algorithms under equal and unequal evolutionary rates. *Molecular Biology and Evolution*, 11(3):459–468, 1994.

Compares parsimony, compatibility, ML, Fitch-Margoliash, and NJ



- [28] S. Kumar, M. Nei, K. Tamura, and M. Nei. Prospects for inferring very phylogenies by using the neighbor-joining method. Technical report, Center for Evolutionary Functional Genomics, The Biodesign Institute, and School of Life Sciences, Arizona State University; Department of Biological Sciences, Tokyo Metropolitan University, 2004. Available for download at <http://www.pnas.org/cgi/reprint/101/30/11030>.

Gives general background information on the neighbor joining algorithm and talks about efficiency of the algorithm found through testing.

- [29] Cecilia Lanave, Giuliano Preparata, Cecilia Saccone, and Gabriella Serio. A new method for calculating evolutionary substitution rates. *Journal of molecular evolution*, 20:86–93, 1984.
- [30] C. Macken, H. Lu, J. Goodman, and L Boykin. The value of a database in surveillance and vaccine selection. In A. D. M. E. Osterhaus, N. Cox, and A. W. Hampson, editors, *Options for the Control of Influenza IV*, pages 103–106. Elsevier Science, Amsterdam, 2001.

This publication describes the Los Alamos National Laboratory Influenza Sequence Database. The database itself is available online at <http://www.flu.lanl.gov/>

- [31] B. Morgenstern. Dialign: Multiple dna and protein sequence alignment at bibiserv. *Nucleic Acids Research*, 32:W33–W36, 2004.

Main reference for DIALIGN.

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

Presents the new method of reconstructing phylogenetic trees, Neighbor Joining, also used a computer simulation to test the efficiency of the new method compared to other methods commonly used.

- [33] C. Notredame, D.G. Higgins, and J. Heringa. T-coffee: A novel method for fast and accurate multiple sequence alignment. *Journal of Molecular Biology*, 302:205–217, 2000.

Introduction to T-Coffee, including description of algorithm.

- [34] P. Nuin, Z. Wang, and E.R.M. Tillier. The accuracy of several multiple sequence alignment programs for proteins. *BMC Bioinformatics*, 7:471, 2006.

Comparing CLUSTALW, Dialign (2.2), Tcoffee, POA, Muscle, MAFFT, ProbCons, Dialign-T, and Kalign

- [35] F. Opperdoes. Construction of a distance tree using clustering with the unweighted pair group method with arithmetic mean (upgma), 1997.

Provides small examples of the UPGMA algorithm.

- [36] F. Opperdoes. The neighbor-joining method, 1997. Available for download at <http://www.icp.ucl.ac.be/~opperd/private/neighbor.html>.

Presents information of Neighbor Joining programs that are available, information on negative branch lengths, and gives small example of how the algorithm works.

- [37] A. Pang, A. Smith, P. Nuin, and E.R.M. Tillier. Simprot: Using an empirically determined indel distribution in simulations of protein evolution. *BMC Bioinformatics*, 6:236, 2005.

SIMPROT tool for generating sequences, currently available via Tillier's web site at <http://www.uhnres.utoronto.ca/labs/tillier/software.htm>.

- [38] W. Pearson, G. Robins, and T. Zhang. Generalized neighbor-joining: More reliable phylogenetic tree reconstruction. Technical report, Department of Computer Science, University of Virginia, Department of Biochemistry, University of Virginia, 2004. Available for download at [http://www.cs.virginia.edu/papers/gnj\\_final.pdf](http://www.cs.virginia.edu/papers/gnj_final.pdf).

Presents background information on the original Neighbor-Joining method and then compare it to a newer Neighbor Joining Method the Generalized Neighbor Joining Method.

- [39] M. Pérez-Losada, M.L. Porter, L. Tazi, and K. Crandall. New methods for inferring population dynamics from microbial sequences. *Infection, Genetics, and Evolution*, 7(1):24–43, 2007.

Good, current overview of the MSA world.

- [40] D. Robinson and L. Foulds. Comparison of phylogenetic trees. *Mathematical Biosciences*, 53:131–147, 1994.

This paper introduces the method of symmetric difference distance between the topologies of two phylogenetic trees, commonly known as the Robinson-Foulds distance.

- [41] D.F. Robinson and L.R. Foulds. Comparison of phylogenetic trees. *Mathematical Biosciences*, 53:131–147, 1981.

Describes the Robinson-Foulds metric for comparing trees, in gnarly detail.

- [42] F. Rodríguez, J. L. Oliver, A. Marín, and J. R. Medina. The general stochastic model of nucleotide substitution. *Journal of theoretical biology*, 142:485–501, 1990.

- [43] C.A. Russo, N. Takezaki, and M. Nei. Efficiencies of different genes and different tree-building methods in recovering a known vertebrate phylogeny. *Molecular Biology and Evolution*, 13(3):525–536, 1996.

Compares NJ, minimum evolution, maximum parsimony, ML

- [44] N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4:406–425, 1987.

This paper introduces the method of neighbor-joining.

- [45] J.M. Sauder, J.W. Arthur, and R.L. Dunbrack Jr. Large-scale comparison of protein sequence alignment algorithms with structure alignments. *Proteins*, 40:6–22, 2000.

Reference for the Modeler score.

- [46] J. Stoye, D. Evers, and F. Meyer. Rose: generating sequence families. *Bioinformatics*, 14(2):157–163, 1998.

Program for generating test sequence data and known trees

- [47] Jens Stoye, Dirk Evers, and Folker Meyer. Rose: generating sequence families. *Bioinformatics*, 14(2):157–163, 1998.
- [48] S.J. Sul and T.L. Williams. A randomized algorithm for comparing sets of phylogenetic trees. Technical report, Department of Computer Science, Texas A&M University, 2006. Available for download at <http://www.cs.tamu.edu/academics/tr/tamu-cs-tr-2006-9-3>.
- [49] K. Tamura and M. Nei. Estimation of the number of nucleotide substitutions in the control region of mitochondrial dna in humans and chimpanzees. *Molecular biology and evolution*, 10(3):512–526, 1993.
- [50] J.D. Thompson, D.G. Higgins, and T.J. Gibson. Clustalw: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22:4673–4680, 1994.

Primary ClustalW reference

- [51] J.D. Thompson, F. Plewniak, and O. Poch. Balibase: A benchmark alignments database for the evaluation of multiple sequence alignment programs. *Bioinformatics*, 15(1):87–88, 1999.
- [52] J.D. Thompson, F. Plewniak, and O. Poch. A comprehensive comparison of multiple sequence alignment programs. *Nucleic Acids Research*, 27(13):2682–2690, 1999.

Ten alignment programs compared using BALiBASE benchmark alignment database.

- [53] C. Tseng. Cmsc 838t - bioinformatics and high performance computing. Lecture, University of Maryland, College Park, 2004. Available for download at <http://amber.cs.umd.edu/class/838-s04/lec5.pdf>.

Presents Phylogenetics, Phylogenetic Trees, and different methods of producing the trees.

- [54] Various. Www-servers of felsenstein lab. Available for viewing at <http://evolution.genetics.washington.edu/>.

A website containing a collection of work done by members of the Felsenstein lab at the University of Washington, Seattle.

- [55] L. Wall, T. Christiansen, and J. Orwant. *Programming Perl (3rd Edition)*. O'Reilly, 2000.

The classic Perl book. If you buy only one Perl book, buy this one.

- [56] Ziheng Yang. Maximum-likelihood estimation of phylogeny from dna sequences when substitution rates differ over sites. *Molecular biology and evolution*, 10(6):1396–1401, 1993.

- [57] Ziheng Yang. Maximum likelihood phylogenetic estimation from dna sequences with variable rates over sites: approximate methods. *Journal of molecular evolution*, 39:306–314, 1994.

- [58] D.J. Zwickl. *GARLI manual (version 0.95)*. World Wide Web, 2006.

GARLI may be downloaded at <http://www.bio.utexas.edu/faculty/antisense/garli/Garli.html>.

- [59] D.J. Zwickl. *Genetic algorithm approaches for the phylogenetic analysis of large biological sequence datasets under the maximum likelihood criterion*. PhD thesis, University of Texas at Austin, Austin, Texas, 2006.

GARLI may be downloaded at <http://www.bio.utexas.edu/faculty/antisense/garli/Garli.html>.