SOFTWARE SIMULATION OF PHYLOGENETIC TREE BASED ON MIXED AND COMPLEX MODELS

A

PROJECT

Presented to the faculty of University of Alaska Fairbanks

In Partial Fulfillment of the Requirements of

MASTERS IN SOFTWARE ENGINEERING

By

Md. Muksitul Haque

Fairbanks, Alaska

August 2008

SOFTWARE SIMULATION OF PHYLOGENETIC TREE BASED ON MIXED AND COMPLEX MODELS

By

Md. Muksitul Haque

RECOMMENDED:

_____

Advisory Committee Co-chair Date

_____

Advisory Committee Co-chair Date

_____

Advisory Committee Member Date

APPROVED:

_____

Dept Head, Computer Science Department Date

_____

Dean, College of Science, Engineering, and Mathematics Date

_____

Dean of the Graduate School Date

_____

Date

ABSTRACT

The aim of this project was to create a software system called PhyloSim which can generate complex data from a synthetic phylogeny. This new system uses multi or mixed mathematical models and variable input parameters for construction of phylogenetic trees. Models that were implemented include covarion, mixture-models, GTR+Γ, GTR+Γ+I, F84, HKY (K2P, F81 and JC69), JTT, WAG, PAM, BLOSUM, MTREV and CPREV. Many of these models are discussed in this report.

These more complex mixture models are of interest because they are believed to be more biologically realistic than those in widespread use currently. However, in order to understand their performance in phylogenetic inference it is necessary to simulate data according to them and see how successful inference schemes are in recovering the tree. Currently available simulators are very limited and insufficient for studying newer phylogenetic models. PhyloSim is an improvement in this respect.

PhyloSim was developed in accordance with a conventional software development plan. The software development approach uses a hybrid of a classical software engineering approach and an agile programming approach. The agile part entails the sequential development of a series of phylogenetic models. Each model was tested carefully before then next was started. The developments were done in accordance with good software engineering practices (i.e. requirement analysis, followed by design, implementation and test). This approach and its results are described in the report and its appendices. A portion of this source code is included in an appendix. Further software evolution is expected to be based on results from the future use of PhyloSim.

ACKNOWLEDGEMENTS

I thank Dr. Peter J. Knoke, Department of Computer Science, University of Alaska Fairbanks, Dr. John Rhodes, Department of Mathematics, University of Alaska Fairbanks and Dr. Orion Lawlor, Department of Computer Science, University of Alaska Fairbanks for the guidance and support they provided.

I would also like to express my sincere appreciation to Dr. John Rhodes, Associate Professor of Mathematics and Dr. Elizabeth S. Allman, Associate Professor of Mathematics, University of Alaska Fairbanks for suggesting this study and for their significant contribution to the domain-specific research. I would also like to thank Dr. Allman for the time she took for extensive testing of all the models and for performing simulations with a broader range of parameter values and trees.

Finally I would like to thank people in The UAF Biotechnology Computing Research Group especially, James Long and Shawn Houston, for their support. This work was partially supported by AK INBRE Grant Number 5P20RR016466 from the National Center for Research Resources (NCRR), a component of the National Institutes of Health (NIH). It was also supported by the National Science Foundation, through DMS grant #0714830.

TABLE OF CONTENTS

CHAPTER 1. INTRODUCTION

The goal of this project was to create a flexible software system for simulating the evolution of biological sequences, such as DNA and proteins. This software incorporates mixture-models[1][2][3], covarion[4][5][6][7] models and other complex models of current interest in phylogenetics. It is intended to be useful for assessing the performance of phylogenetic inference schemes[8][9][10][11]. These more complex mixture models are of great current interest, as they are believed to be more biologically realistic than those in widespread use. Currently available simulators are very limited, and are insufficient for studying newer models.

Popular phylogenetic sequence simulators such as Seq-Gen[12][13] were studied, and a system of greater flexibility for the models used for such program and the their output trees was proposed. The main feature of the proposed system was a general framework in which any of the currently used model information can be entered. Using this information, the system simulates evolution along a phylogenetic tree and produces sequences for the leaves and internal nodes of the tree. The above issues are further discussed in chapters 2 and 3.

The proposed flexible system was developed and subsequently named PhyloSim. The implemented mathematical models for PhyloSim include DNA, protein and codon models. Specifically, the models have been implemented: are GTR, GM[14], F81, F84, HKY, JC, K2P, TN, CFN, GTR+Γ, GTR+Γ+I, TreeMixture, mixture, Invariable, BLOSUM, CPREV, DAYHOFF, DCMUT, JTT, MTMAM, MTREV, VT, WAG, WAGSTAR, EqualDistCovarion and ScaledCovarion[15][16][17]. These models are discussed further in chapters 3 and 7. Appropriate data structures were used to store the information of nodes and edges as the tree is being constructed. All of the models are rigorously tested with varying input and were checked against other popular phylogenetic inference[18][19] software such as PAUP*[20] and PhyML[21] for consistency. Test methods and results are discussed in chapter 8.

PhyloSim takes required model parameters from an input file and saves the output as output.nex file. An advantage of the PhyloSim is that it accepts custom bases, and flexible tree and matrix sizes can be used. There are options for recording ancestral sequences and displaying sequences in interleaved or non-interleaved formats. Multiple simulations or datasets can be created in the same run of the program. Complex model construction supports across-site rate variation & covarion models, mixture models allow multiple models to be used in different ratios on the gene sequence along the phylogenetic tree. Substitution rates are one of the most fundamental parameters in a phylogenetic analysis and are represented in phylogenetic models as the branch lengths on a tree. Variation in substitution rates across an alignment of molecular sequences is well established and likely caused by variation in functional constraint across the genes encoded in the sequences. Rate variation across alignment sites is important to accommodate in a phylogenetic analysis; failure to account for across-site rate variation can cause biased estimates of phylogeny or other model parameters. Traditionally, rate variation across sites has been modeled by treating the rate for a site as a random variable drawn from some probability distribution (such as the gamma probability distribution) or by partitioning sites to different rate classes and estimating the rate for each class independently[22]. Tree mixtures allow changes according to different trees simulated by different models but with the same taxa in each of them.

The GUI created by gtk+[23] library includes an easy to use tree creator. The GUI comes with a template for each of the models; the model user can make use of these to create the input file for any of the model data very quickly. It also allows the user to see the output below the input window. The GUI comes with other options and a help menu.

The development of PhyloSim was a complex project because it requires the use of great deal of data, because it uses a different style of programming approach, and because of its functional and non-functional requirements. To implement the project successfully with high quality software, a hybrid software engineering approach has been used.

These software engineering issues and methods are further discussed in chapter 4 through 8. Chapter 3 and 7 contains flowcharts and algorithms of the architecture, design and implementation of the PhyloSim software and the mathematical models.

Finally, chapter 9 provides a summary and conclusions based on results from the project, which chapter 10 provides recommendation for further work on PhyloSim development. Such further work is expected to be based on findings from its further use. Increased use could be expected if access were improved by making it web-based. Also larger models could be supported with continuing good performance by use of parallel programming, better mathematical models and better algorithms.

CHAPTER 2. DOMAIN BACKGROUND AND RELATED WORK

2.1 Introduction

In modern molecular biology, new sources of data are present today. Biological sequences such as DNA and protein retain similarity to their ancestral parents. Mathematical methods can be used for analyzing the similarity and the differences in the sequences to infer phylogenies. Through the aid of mathematical thinking in biology, we nowadays have several tools to extract evolutionary information from sequence data. But still challenges remain to improving methods, and such research is ongoing.

2.2 Molecular Evolution

Natural selection is the main mechanism through which evolution occurs. For selection to occur however there must be underlying changes in genetic makeup within a species. Because selection acts to reduce variability, new sources of genetic variation are introduced at the molecular level, such as the DNA of each individual, through random mutation.

With changes in their DNA, some offspring might be more or less capable of living than their parents. A particular gene's DNA might mutate over generations and become very different than its ancestral form. So many species descending from the same ancestor can have different DNA forming the same gene. The similarity shows common ancestors while difference shows evolutionary divergence.

So we can conclude that species with more similar genetic sequences are probably more closely related.

But for inferring an evolutionary tree relating a large number of different species with varying degree of similarity of a chosen gene, we need more elaborate mathematical ideas of how the mutations occurred.

First let us cover some biological background.

2.2.1 DNA structure

The structure of DNA is a double helix, with about 10 nucleotide pairs per helical turn. Each spiral strand, composed of a sugar phosphate backbone and attached bases, is connected to a complementary strand by hydrogen bonding (non- covalent) between paired bases, adenine (A) with thymine (T), and guanine (G) with cytosine (C). Adenine and thymine are connected by two hydrogen bonds (non-covalent) while guanine and cytosine are connected by three. This structure was first described by James Watson and Francis Crick in 1953.

Because of chemical similarity, adenine and guanine are called purines, and cytosine and thymine are called pyrimidines. We always find A paired with a T or G paired with a C. So knowing one side of the ladder helps to know the other side. For example if we have sequence

AATTGGCC

Then the complementary sequence would be

TTAACCGG

Some sections of this DNA are used to form genes that have information for creation of proteins. Three consecutive bases in these genes create codons, where each codon specifies a specific amino acid to be placed according to genetic code. There are 4^3=64 different codons and 20 amino acids. Three of the codons signal the end of the protein sequence, and it do not code for an amino acid.

Not all DNA are coded into genes, 97% of human DNA is believed to be non-coding. Some of this may be meaningless, but other parts probably serve as controllers of some sort. It is not yet understood.

2.2.2 Mutations

A common mutation in copying sequences of DNA is called a base substitution. This happens by replacing one base by another base at a certain site in the sequence. An example of a base substitution is

AATTGGCCC

ATTTGGCCC

A base substitution of A→T has happened on the 2^{nd} site.

In a base substitution if a purine replaces a purine or a pyrimidine replaces a pyrimidine then it is called transition; an interchange of these classes is called a transversion. Transitions usually occur more than transversions because the chemical structure of the molecule changes less under a transition than a transversion.

Special cases of base substitution are hidden mutation such as C→T→ G in which a subsequent mutation hides an earlier mutation. This may happen in the case where we do not have all the sequences of all generations, and we would not know. Back mutation C→T→C is a special case of hidden mutation.

Other things that happen more rarely in natural populations are insertion or deletion of one or consecutive bases.

2.2.3 Aligned Orthologous Sequences

All parts of the genome are believed to be descended from one much smaller ancestral piece of nucleic acid. All genomes have originated from this ancestral sequence by gene duplication, loss of parts of resulting genomes, insertion and rearrangements of various sorts. As complete genomes have been sequenced, both gene family and genome structure have become available in

those species. Specialized methods are required to gain an understanding of the events in gene family and genome evolution and to assist in inferring phylogenies.

One such method is sequence alignment. In order to compare two or more sequences, it is required to align the conserved and unconserved residues across all the sequences. The residues form a pattern from which the relationship between sequences can be determined with phylogenetic programs. When the sequences are aligned, it is possible to identify locations of insertions or deletions because of their divergence from their common ancestor. There are three possibilities:

1. The bases match: this means that there is no change since their divergence, although back mutation is possible
2. The bases mismatch: this means that there is a substitution since their divergence.
3. There is a base in one sequence, no base in the other: there is an insertion or a deletion since their divergence.

A good alignment is important for the construction of phylogenetic trees. The alignment will affect the distances between two different species and this will influence the inferred phylogeny[16].

There are some good search algorithms such that when given a gene identified in one organism it can locate similar genes in related organisms. By experimentally verifying that these in fact are genes and they have similar functions, we can assume that they are orthologous, which means they came from a common ancestral sequence.
For some data, we can align orthologous sequences from different organisms easily. For others, finding good alignment is difficult; with large variation among sequences even the best software may have a hard time aligning them. In particular, if many insertions or deletions have occurred, alignment can be difficult. So, a mix of algorithm and ad hoc human adjustment is often used for producing better results.
Once aligned orthologous sequences are in hand, the next goal is to produce a phylogenetic tree that describes their likely decent from a common ancestral sequence.

2.2.4 Newick Format[24] for phylogenetic tree representation

The Newick standard for representing trees in computer readable form makes use of trees and nested parenthesis. It was created by in 1857 by English mathematician Arthur Cayley.

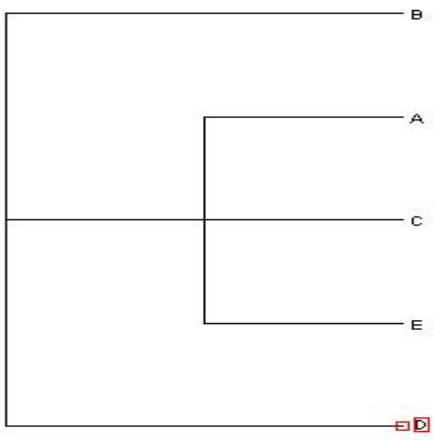


Fig-1: graphical representation of (B,(A,C,E),D);

An example is given in Figure 1 the tree shown is represented by the following sequence of characters
(B,(A,C,E),D);

Here the tree ends with a semicolon. The bottom most node (shown at the left in Figure 1) is an interior node. Biologically, this represents the most recent common ancestor and is called the root of the tree. Interior nodes are represented by a pair of matched parenthesis. In between them are the nodes that are immediately descendents of that node, separated by comma.

In the above tree the immediate descendants are B, another interior node and D. Other interior node is given by a pair of parenthesis, enclosing representation of its immediate descendants, A, C, & E. In general there can be many interior nodes and results will be further nesting of parenthesis, to any level.
Leaves are represented by names; such as A,B,C,D,E in the figure, a name can be any string of printable characters, except blanks, colons, semicolons, parenthesis and square brackets.

Also trees can multifurcate at any level. We can add branch length into a tree by putting a real number with or without decimal point, after a node and proceeded by a colon. This represents the length of the branch immediately below that node. For Example

((((A:0.01,B:0.02):0.03,C:0.04):0.05,D:0.06):0.07,Q:0.08);

is graphically represented using Dendroscope[25] as shown in Figure 2.

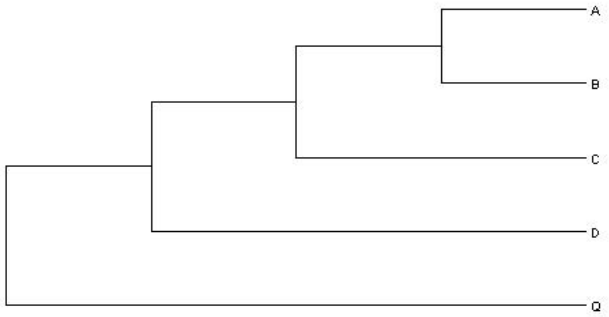


Fig-2: graphical representation of ((((A:0.01,B:0.02):0.03,C:0.04):0.05,D:0.06):0.07,Q:0.08);

A tree starts on the first line, and bigger trees with more branches can continue to subsequent lines. Blanks can be entered anywhere, except in the middle of a species name or a branch length.

Examples of Newick trees are

(((A:0.01,B:0.02):0.03,C:0.04):0.05,D:0.06);
(((A:1, B:2):3, C:4):4, ((D:1, E:2):3, F:4):4);
(A:0.01,B:0.02);
(Bovine:0.69395,(Gibbon:0.36079,(Orang:0.33636,(Gorilla:0.17147,(Chimp:0.19268,Human:0.11927):0.08386):0.06124):0.15057):0.54939,Mouse:1.21460):0.10;

Although both binary and non-binary trees are used, by using edge lengths of 0, we can represent non-binary trees in Newick format as if they are binary.
There are some limitations for the representation of the Newick tree. One of them is the left right order of desendants of a node affects the representation, even though it is not biologically meaningful. So

(A,(B,C),D); is same as (A,(C,B),D);

Also, in phylogeny inference we generally cannot infer the position of the root. We represent the tree as unrooted tree. When describing inferences of such cases

(B,(A,D),C); is same as unrooted tree (A,(B,C),D)

Although some limitations do exist for the readability of Newick representations, this standard is in widespread use in phylogenetics.

2.3 The Software

We need to use software for inferring phylogenies from molecular data. Although new software packages for this purpose are being written every day, two general phylogenetics inference packages are in widespread use: they are PAUP* and PHYLIP.

Both of these use mathematical models of the substitution process in their approaches to statistical inference of phylogenetic trees. Seq-Gen is a program that uses some of these models to simulate sequence evolution. It is an important tool for understanding how inference works, since it is almost impossible to produce experimental datasets where the phylogenetic tree is known beyond any doubt. Use of simulated data sets with known phylogenetic trees can be used to test how well inference can be performed.
Many phylogenetics packages use nexus file formats and there are packages that export data to other formats as well.

Brief descriptions of the packages are given as follows:

PAUP*: Phylogenetic Analysis using parsimony and other methods. David Swofford's package is probably the most widely used package for the inference of evolutionary trees[11][21].

PHYLIP: PHYLIP is a free package of programs for inferring phylogenies. It is distributed as source code, documentation files, and a number of different types of executables. Written by Joe Felsenstein of the Department of Genome Sciences and the Department of Biology at the University of Washington [8].

Seq-Gen: Seq-Gen is a program that simulates the evolution of nucleotide or amino acid sequences along a phylogeny, using common models of the substitution process. A small range of models of molecular evolution are implemented including the general reversible model. State frequencies and other parameters of the model may be given and site-specific rate heterogeneity may also be incorporated in a number of ways. Any number of trees may be read in and the program will produce any number of data sets for each tree. Thus large sets of replicate simulations can be easily created. Seq-Gen is created by Andrew Rambaut & Grassly NC[12][13].

CHAPTER 3. DEFINITION OF THE MODELS PROBLEM

3.1 Probabilistic Models of DNA mutation

Mutations are not always rare and hidden and back mutations may occur often. Trying to mathematically describe the mutation process that DNA undergoes as evolution proceeds is important. That is the only way we can gain insight into things we cannot observe.

We need mathematical models that describe how mutations happen. Since there is a probability in this happening, we describe mutations probabilistically.

3.1.1 A simple example:

We shall begin our modeling approach with a basic example.

Let us imagine a tree with only one edge, we will model one site in a DNA sequence from ancestral to descendant sequence. We will focus on how the sequences change classes. Here R=purine, Y=pyrimidines

In our example we have ancestral and descendant sequence

Suppose we somehow had access to an ancestral sequence S0 and a descendant sequence S1.

S0 : RRYRYRYYRYYYRYRYYRRYY
S1 : RYYRYYYYRYYYRYRYYRRYR

To represent a site in an ancestral sequence we specify the probabilities that site may be occupied by R or Y.

Here Pr, Py =0.5,0.5 would indicate an equal chance of each while Pr, Py =0.6,0.4 would indicate a greater likelihood of purine. But the probabilities should always add to 1. Since this is a root distribution of the sequence, where the two states are R,Y.

In case of General Markov (GM) matrix it must have at least 2x2 matrix where each element represent the probability of change

$$\begin{pmatrix} R\rightarrow R & R\rightarrow Y \\ Y\rightarrow R & Y\rightarrow Y \end{pmatrix}$$

For example, if we compare it with the sequence in S0 and S1, we would get the following data

$P_{R\rightarrow R} \approx \frac{7}{9}$ $P_{R\rightarrow Y} \approx \frac{2}{9}$

$P_{Y\rightarrow R} \approx \frac{1}{11}$ $P_{Y\rightarrow Y} \approx \frac{10}{11}$

Here $P_{R\to R}$ indicates the probability of R base becoming R and $P_{R\to Y}$ indicates the probability of R becoming a Y for the sequence (where 9 is the total number of events and 2 & 7 are the number of those particular events). So we can see that adding them we always get 1.

3.1.2 A continuous-time version

It is natural to think mutation as occurring at discrete times for an evolving organism. But the generation duration is usually quite small on an evolutionary time scale, so it is common to use continuous time.

For this reason, we use certain rates at which the different mutations occur and then we organize them into a matrix such as

$$Q = \begin{pmatrix} q_{RR} & q_{RY} \\ q_{YR} & q_{YY} \end{pmatrix}$$

So here, q_{RY} denotes instant rates at which Y replaces R states and is measured in units like substitution per site/year. The entries in each row of Q must add to give 0 and non-diagonals must be positive while diagonals must be negative.

So we have a system of differential equation

$\frac{d}{dt} p_R(t) = p_R(t)q_{RR} + p_Y(t)\, q_{YR}$

$\frac{d}{dt} p_Y(t) = p_R(t)q_{RR} + p_Y(t)q_{YY}$

We can write this differential equation in matrix form as,

$\frac{d}{dt}p_t = p_tQ$ where $p_t = (p_R(t)\ \ p_Y(t))$

Using initial values of p_0 we obtain the solution

$p_t = p_o\exp(Qt)$

We can use the Taylor series formula here, as this formula involves the exponential of a matrix, So for a square matrix A

$$e^A = 1 + \frac{A}{1!} + \frac{A^2}{2!} + \frac{A3}{3!} + \cdots,$$

If A can be diagonalized, $A=S\Lambda S^{-1}$ with Λ the diagonal matrix of eigenvalues of A and S matrix, whose columns are corresponding right eigenvectors

$e^A = Se^{\Lambda}S^{-1}$

where e^{Λ} is a diagonal matrix, with diagonal entries the exponential of entries of Λ

In our model we have $p_t=p_0e^{Qt}$ hence $M(t)=e^{Qt}$.

As we compare sequences from time 0 to time t, the entries of M(t) are conditional probability of various mutation, hence, M(t) is a single matrix describing mutation along an edge representing time t. It is therefore like the matrix M given in section 3.1.1.

In a Markov matrix, M as in section 3.1.1, we describe the mutation process along an entire edge and all the elements of the row sum to 1. In a rate matrix Q the row sum to zero and we describe instantaneous mutation process. We get a Markov matrix by applying the exponential function to a rate matrix times the elapsed time.

3.2.1 Markov models of DNA base substitution

Let us consider a single vertex in a tree that can have four bases (A, G, C, T) . Here purines precede pyrimidines.

Let us take a rooted tree T^p. Then the GM model T^p consists of

1. $p_\rho =(p_A, p_G, p_C, p_T)$ root distribution vector, where sums of all non-negative entries is 1.
2. For each edge we have e=(u,v) a 4x4 Markov matrix M_e, in which each row sums to 1 and values are non-negative
   So we have $M_{ij}=P(S_v=j|S_u=i)$

$$M_e= \begin{pmatrix} p_{AA} & p_{AG} & p_{AC} & p_{AT} \\ p_{GA} & p_{GG} & p_{GC} & p_{GT} \\ p_{CA} & p_{CG} & p_{CC} & p_{CT} \\ p_{TA} & p_{TG} & p_{TC} & p_{TT} \end{pmatrix}$$

Here the only mutations allowed are base substitutions.

3.2.2 A common rate-matrix variant of the Model

In order to specify mutation process along the various edges of the tree we create a continuous time 4x4 rate Matrix Q, whose off diagonal entries are non negative and rows add to 0

$$Q = \begin{pmatrix} q_{AA} & q_{AG} & q_{AC} & q_{AT} \\ q_{GA} & q_{GG} & q_{GC} & q_{GT} \\ q_{CA} & q_{CG} & q_{CC} & q_{CT} \\ q_{TA} & q_{TG} & q_{TC} & q_{TT} \end{pmatrix}$$

with q_{ij} the rate per site that i is replaced by j

2b) if edge lengths $t_e \geq 0$ are given for all edges in T^p

the Markov Matrix

$M_e = M(t_e) = e^{Qte}$

Can be interpreted as in (2) of 3.2.1 above for the edge e, directed away from the root.

3.2.3. Properties of Markov Matrix (M)

1) A Markov matrix always has $\lambda_1 = 1$ as its largest eigenvalue and all eigenvalues $|\lambda| \leq 1$ one eigenvector corresponding to λ_1 has all non-negative entries

2) A Markov matrix with all entries positive has largest eigenvalue 1 with multiplicity 1 and the one eigenvector associated to $\lambda_1 = 1$ has all positive entries.

3.3 Jukes Cantor and Kimura models

3.3.1 Jukes Cantor

Jukes Cantor adds to GM that all bases occurs with equal probability in ancestral sequence

$p_o = (¼, ¼, ¼, ¼)$

In the Jukes Cantor model, we use transition matrices of the form

$$M = \begin{pmatrix} 1-a & a/3 & a/3 & a/3 \\ a/3 & 1-a & a/3 & a/3 \\ a/3 & a/3 & 1-a & a/3 \\ a/3 & a/3 & a/3 & 1-a \end{pmatrix}$$

where a larger value of a denotes more mutation.

Equivalently we use the rate matrix

$$Q = \begin{pmatrix} -1 & 1/3 & 1/3 & 1/3 \\ 1/3 & -1 & 1/3 & 1/3 \\ 1/3 & 1/3 & -1 & 1/3 \\ 1/3 & 1/3 & 1/3 & -1 \end{pmatrix}$$

3.3.2 Kimura Models

Jukes Cantor uses a single parameter a to denote mutation, but Kimura may add more parameters per edge.

Kimura 2 parameter adds different probabilities of transition and transversion.

$$M = \begin{pmatrix} * & b & c & c \\ b & * & c & c \\ c & c & * & b \\ c & c & b & * \end{pmatrix}$$

Where b =transition and c=transversion and *=1-b-2c
A continuous version uses a rate matrix

$$Q = \begin{pmatrix} * & \beta & \delta & \delta \\ \beta & * & \delta & \delta \\ \delta & \delta & * & \beta \\ \delta & \delta & \delta & * \end{pmatrix}$$ where $* = -\beta - 2\delta$

Kimura assumes the root distribution vector is
$p_o = (1/4, 1/4, 1/4, 1/4)$

3.4 General Time Reversible Model

The continuous time version of the Jukes Cantor and Kimura Models are time – reversible. For example in an edge with ancestral and descendant, if we reverse the flow of time, we can describe the evolution with exactly the same parameters.
If p is the ancestral base distribution, M is the Markov matrix and P is the joint distribution of bases in ancestral and descendant sequence, then
$P(i,j) = p_i M(i,j)$, or as a matrix equation
$P = \text{diag}(p)M$.

Thus time reversibility means $P = P^T$ or

$$\text{diag}(p)M = (\text{diag}(p)M)^T = M^T \text{diag}(p)$$

So, for time-reversibility to hold we need,

$$\text{diag}(p)Q = Q^T \text{diag}(p)$$

If $p = (p_A \;\; p_G \;\; p_C \;\; p_T)$ is the root distribution & we use a rate matrix

$$Q = \begin{pmatrix} * & p_G\alpha & p_C\beta & p_T\gamma \\ p_A\alpha & * & pc\delta & p_T\epsilon \\ p_A\beta & p_G\delta & * & P_T\eta \\ p_A\gamma & P_G\epsilon & P_C\eta & * \end{pmatrix}$$

where rows sum to zero and where the relative rates $\alpha, \beta, \delta, \epsilon, \eta, \gamma \geq 0$, then we use a time reversible model.

Note that $\vec{P}Q = \vec{O}$, so P is an eigenvector of Q with eigenvalue O.

The General Time Reversible (GTR) assumes as parameters
1) GTR rate matrix Q on all edges
2) Scalar edge length
3) Root base distribution that is stable, p is an eigenvector of Q of eigenvalue 0

Nice feature of GTR

GTR is the most general neutral, independent, finite-sites, time-reversible model possible. It was first described in a general form by Simon Tavaré in 1986

1) Time reversibility ignores location of root, reducing search time for optimal trees
2) Common rate matrix for all edges means less parameters
3) Common rate matrix also imposes commonality to mutation process over the whole tree, hence may be biologically more realistic

While nature does not necessarily follow any particular models, models with more parameters are intractable for computation with large number of taxa. Its desirable to use models with few parameters provided they capture the key aspect of reality. Having a small number of parameters also avoids overfittting data while keeping variance of inferred trees lower.

3.5 A simple construction of a model

Example GM model

The GM model takes input matrices from the user; the user can enter any number of matrices and the required root parameters for the model. The GM model simulates the phylogenetic tree and outputs the DNA sequence for each of the leaves.

Example of an input tree would be ((1:[M],2:[N]):[M1],((4:[P1],5:[P2]):[P3],3:[P4]):[L]);

where 1 to 5 are different leaf numbers and M,N,M1,P1,P2,P3,P4 and L are the matrices for each of the edge. The system allows for input of Markov matrices for each edge. The program returns sequences at the leaves of the tree that have been generated according to such parameters.

For construction of the tree the user must specify the root of the tree. There is a "root distribution" parameter that gives the probabilities that the root is in each of the possible states (four for DNA). For each edge there is a transition matrix giving the transition probabilities of various substitutions along that edge. The form of the matrix depends on the model of substitution chosen.

A sample input file for GM

A sample input file that is taken by our program is given below

```
model = GM
tree = (((1:[P1],2:[P2]):[M1],(3:[P3],4:[P4]):[L]):[M],5:[N]);
states = DNA
root = [0.2,0.3,0.2,0.3]
nchar = 200000
P1 = {0.7,0.1,0.1,0.1,0.1,0.7,0.1,0.1,0.1,0.1,0.1,0.7,0.2,0.1,0.1,0.1,0.1,0.7};
P2 = {0.6,0.2,0.1,0.1,0.1,0.1,0.7,0.1,0.1,0.1,0.2,0.1,0.1,0.6,0.1,0.1,0.1,0.1,0.1,0.7};
M1 = {0.6,0.2,0.1,0.1,0.1,0.2,0.6,0.1,0.1,0.1,0.1,0.1,0.1,0.6,0.2,0.1,0.1,0.2,0.1,0.1,0.6};
P3 = {0.6,0.2,0.1,0.1,0.1,0.1,0.7,0.1,0.1,0.1,0.1,0.1,0.1,0.7,0.1,0.1,0.1,0.1,0.2,0.6};
P4 = {0.7,0.1,0.1,0.1,0.1,0.1,0.7,0.1,0.1,0.1,0.1,0.1,0.1,0.7,0.1,0.1,0.1,0.1,0.1,0.7};
```

L = {0.7,0.1,0.1,0.1,0.1,0.7,0.1,0.1,0.1,0.1,0.7,0.1,0.1,0.1,0.1,0.7};
M = {0.6,0.1,0.2,0.1,0.2,0.6,0.1,0.1,0.1,0.1,0.6,0.2,0.1,0.2,0.1,0.6};
N = {0.7,0.1,0.1,0.1,0.1,0.6,0.1,0.2,0.1,0.1,0.7,0.1,0.1,0.1,0.2,0.6};

Here, the model name is given as GM. The tree is in Newick[24] format where the capital letters inside the brackets "[" and "]" indicate matrix names and the numbers are taxa names. After that we have the root parameters, the probability of the four bases in the root DNA sequence, then we have the total length of the sequence, and last is given the edge matrix of each of the edges, this matrix gives the probability of one base changing to same or other base along the edge. Those are 4x4 matrices, whose 16 entries are listed by consecutive rows.

Example GTR model

For the GTR and GTR-like models, the software takes the tree, root-parameters, relative rates, transition-transversion ratio, RYratio. The output of the file is produced as output.nex. It contains DNA bases of the leaves of the phylogenetic tree. The "rate matrix" is part of what is known as a "GTR" model.

The output of the phyogenetic program (output.nex) was fed into the PAUP* program and it produces the input tree to our program for the GTR model, as well as very close approximations of all other parameters. So the GTR model is producing correct result. The Complexity Cluster service of BCRG, UAF was used to make use of PAUP* and check the GTR model for varying input.

For the models all the information is read in from a file for phylogenetic tree construction. The file also contains the number of sequences, so when simulating the tree the input and the output DNA sequence can be any bases (A,C,G,T) long. It also supports Amino acid(20) and binary(0,1) bases. In addition custom bases can be created and flexible input and matrix sizes be used.

The software has been written in C language in Linux environment (gcc compiler used). gsl[26] library has been used for some of the matrix computations, such as finding the exponential; of a symmetric matrix in the GTR model. The GUI was created using GTK+/glade interface designer.

A sample input file for GTR

A sample input file that is taken by our program is given below

model=GTR
tree=((((A:0.01,B:0.02):0.03,C:0.04):0.05,D:0.06):0.07,Q:0.08);
states = DNA
root=[0.4, 0.2, 0.2, 0.2]
nchar=200000
relative-rates = [1, 2, 3, 1, 2, 1]

Here, the model name is given as GTR. The tree is in Newick format where the capital letters indicate leaf names and the numbers edge length, after that we have the root parameters, the probability of the four bases in the root DNA sequence. Then we have the total length of the sequence and last is given the relative rates to construct the rate matrix Q for the GTR model, as in section 3.4.

3.6 Rate Variation Model

In our past examples, Markov models of DNA mutation all assumed that all sites behave identically. But in Biology this is not realistic. It is desirable to model mutation processes so that some sites mutate quickly, other slowly and others not at all.

Invariable sites Model

For creating rate variation models, we can create two classes of sites. The first class (variable sites) mutates but the 2nd class (invariable sites) do not mutate.

If, we examine data from such a model, if we observe mutation at a site, then it must be in the first class. If no mutation is observed, it may be in either class, since it may have been variable but simply never changed, or perhaps had a hidden mutation.

Such a model is GM+I, We formulate this for characters with K states, trees relating n taxa.

For rooted tree, T relating the n taxa

P_ρ is a root distribution vector with k entries for the variable sites

$\{M_e\}_{e \in E(T)}$ a kxk Markov matrix for each edge

r is class size parameter, where r gives the probability a site is variable and 1-r the probability it is invariable.

q is another distribution vector for invariable sites

We can implement GM+I in the PhyloSim program as a mixture model. The input file for GM+I is given below

```
model = Mixture
tree = (((A:0.01, B:0.02):0.03, C:0.04):0.05, D:0.06);
states = DNA
nchar = 2000
nclasses = 2
ClassProportions = [0.6,0.4]

begin-class
   model = GM
   root = [0.1, 0.2, 0.3, 0.4]
   A = {0.7,0.1,0.1,0.1,0.1,0.7,0.1,0.1,0.1,0.1,0.7,0.2,0.1,0.1,0.1,0.7};
   B = {0.6,0.2,0.1,0.1,0.1,0.7,0.1,0.1,0.2,0.1,0.6,0.1,0.1,0.1,0.1,0.7};
   C = {0.6,0.2,0.1,0.1,0.2,0.6,0.1,0.1,0.1,0.1,0.6,0.2,0.1,0.2,0.1,0.6};
   D = {0.6,0.2,0.1,0.1,0.1,0.7,0.1,0.1,0.1,0.1,0.7,0.1,0.1,0.1,0.2,0.6};
end-class

begin-class
   model = Invariable
   root = [0.4,0.3,0.2,0.1]
```

end-class

or we can implement GTR+GAMMA (using GTR instead of GM) as given below

A sample input file for GTR+GAMMA

A sample input file that is taken by our program is given below

```
model=GTR+GAMMA
tree=((((A:0.01,B:0.02):0.03,C:0.04):0.05,D:0.06):0.07,Q:0.08);
states = DNA
root=[0.4, 0.2, 0.2, 0.2]
nchar=200
relative-rates = [1, 2, 3, 1, 2, 1]
alpha=0.5
nclasses=4
```

Rates across sites Models

In GM+I model we use two classes, where we let one of them to not allow mutating. In practice models with fewer parameters than a mixture of GM are usually used.

For a fixed tree T, and rate matrix Q, with root distribution vector ρ

scalar edge length $\{t_e\}_{e \in E(T)}$

if we have m classes of sites, we use m scalar rate parameters $\lambda_1\ \lambda_2\ \lambda_{3\ldots}\ \lambda_m$ where a vector r=(r_1,r_2,r_3…r_m) gives the sizes of the classes so $r_1 + r_2 + r_m = 1$

For ith rate class, we will use rate matrix $\lambda_i Q$ on an edge e of the tree we have Markov matrix.

$M_{e,i} = \exp(t_e \lambda_i Q)$

We combine the classes to get the joint distribution for the mixture model

$P = \sum_{i=1}^{m} r_i P_i$

Similarly we can get a continuous distribution of rates

$P = \int_{\lambda} r(\lambda)\ P_\lambda d\lambda$

It is common to use a Γ-distribution of rates

$r_\alpha(\lambda) = \alpha^\alpha / \Gamma(\alpha)\ \lambda^{\alpha-1} e^{-\alpha\lambda}$

The shape parameter α then adds only one parameter to our model, but allows flexibility in distribution for a better maximum likelihood fit.

We can use a mixture model or we can implement GTR instead of GM as GTR+GAMMA+I. A sample input file is given below

A sample input file for GTR+GAMMA+I

A sample input file that is taken by our program is given below

```
model=GTR+GAMMA+I
tree=((((A:0.01,B:0.02):0.03,C:0.04):0.05,D:0.06):0.07,Q:0.08);
states = DNA
root=[0.4, 0.2, 0.2, 0.2]
nchar=1000
relative-rates = [1, 2, 3, 1, 2, 1]
alpha=0.5
nclasses=4
Pinv=0.2
```

The Covarion Model

In evolution, some sites are unable to change because they code for a part of protein that is essential for the organism to live. After some evolution, those proteins might not be necessary. So they are free to change in different parts of the tree. Fitch and Markowitz called the sites that were free to vary at a particular time "covarions" which indicates some characters of switching between free and not free as evolution proceeds across the tree.

For a DNA model, instead of 4 states, we have 8 states

A^{on}, G^{on}, C^{on}, T^{on}, A^{off}, G^{off}, C^{on}, T^{off},

where on means the site can vary and off means currently that site is invariable

So we have additional parameters, an instantaneous rate S_1 at which "on" state switch to "off" state and instant rate S_2 at which "off" state switch to "on" state.

So we get 8x8 rate matrix

$$\tilde{Q} = \begin{pmatrix} Q - s_1 I & s_1 I \\ s_2 I & -s_2 I \end{pmatrix}$$

Where I denotes 4x4 identity matrix.

Now letting

$\sigma_1 = \frac{S2}{S1+S2}$ $\sigma_2 = \frac{S1}{S1+S2}$

$\tilde{p} = (\sigma_1 p, \sigma_2 p)$

$\tilde{p}$ is a stable distribution for $\tilde{Q}$. The rate matrix $\tilde{Q}$ and root distribution vector $\tilde{p}$ form a time reversible model.

So, for any tree T, with root ρ, we have 8 state model with root distribution vector $\tilde{p}$, rate matrix $\tilde{Q}$, edge length $\{t_e\}_{e \in E(T)}$ where Markov matrix $M_e = e^{t\tilde{Q}}$ is assigned to the edge e[15].

Although inside the program different variation of each of the bases is used, so we have a total of 8 bases (on and off for each base) but when we go to the leaves the program only shows the A, C, G, T bases as output.

A sample input file for Covarion

A sample input file that is taken by our program is given below

```
model = ScaledCovarion
states = DNA
nchar = 20
nclasses = 2
ScalingFactors = [1, 0.5]
tree = (((A:0.01, B:0.02):0.03, C:0.04):0.05, D:0.06);
StateRootDistribution = [0.4, 0.3, 0.2, 0.1]
StateRelativeRates = [1, 2, 3, 1, 2, 1]
ClassRootDistribution = [0.75, 0.25]
ClassRelativeRates = [1]
```

CHAPTER 4. PROJECT PLANNING

Some kind of project planning is the initial step for any project. It usually includes determining objectives of the project, scheduling information, obtaining resources of the project and determining main tasks to be completed. Risk analysis is also an important component of most project plans. Here risks are identified and risk monitoring procedures are defined.

4.1 Objective

Software projects that are finished late, over budget and with poor quality software are common. In software engineering this has been called “Software Crisis”. To avoid such problems, a project plan was created for PhyloSim. The main tasks to be completed were identified, with sequences and schedules.

4.2 Resources needed

1. Hardware needed

The system runs fairly fast on most machines, but for higher number of sites recommended systems for the product are as following.

Recommended for running the software, the user should have the minimum following hardware configuration:

800MHz Pentium 3 processor

With 512 MB of RAM

10 MB of free hard disk space

And a display resolution of 1024x768 with 24-bit color (to support the GUI)

2. Operating System needed

The operating system must be running is Linux or MacOSX

The code can be run on Windows98/NT/2000/XP using cygwin software (not tested yet)

3. Tools needed

gcc compiler

Dependencies of the software includes several freely available libraries for download

They are

gsl library

gtk+ library

dependencies of gtk+

glib

atk

pango

cairo

most of the libraries besides gsl are already installed in popular versions of Linux and only required to be installed for compiling the code in Macintosh operating system.

4.3 Risk Analysis

The purpose of risk analysis is to find, evaluate and decrease the impact of foreseeable risks. This helps to ensure that the project can produce high quality software within the predetermined schedule and budget.

Scenarios (Risk discovered from an ATAM analysis [27][28])

1. Invalid data is entered in the input file
2. Invalid tree format used
3. The hard disk of the user crashes
4. The user tries a high mutation rate in one of the nodes
5. The user tries to implement invalid horizontal gene transfer
6. The user tries to save the file in an unsupported file format
7. The GUI crashes while using in a different operating system other than Linux
8. The software update needs to be done automated
9. Process received from the user is put in queue for processing, all process in queue have the same priority

4.3.1 Risk Themes

There are four types of risks outlined in the previous page. They can be categorized as integrity, technical, logical and usability.

Data Integrity Risk: These are items 1,2. These can be addressed by making sure that the system goes through rigorous data checking while the user is providing them and keeping options disabled (In GUI) so that the user cannot enter invalid entry.

Technical Risk: These are items 3,4,8,9. This should cover other system software and hardware to make sure that the system supports PhyloSim properly.

Logical Risk: These are items 4,5. This should cover that by varying some parameters to a certain degree would result in trees and output data that be irrelevant and not useful.

Usability Risk: These are items 6,7,8,9. These can be addresses by properly stating the functional and non-functional requirements and making sure they are implemented properly.

4.4 Schedule

The Work Breakdown Structure (WBS) lists the main project tasks and the PERT shows their sequencing. The schedule for implementing each task considers both priority and resource requirements. The Gantt chart is for basic scheduling and tracing tool for the project. It was modified whenever needed depending on the progress of the project. For example changing the way tree data is represented in GM model.

4.5 Project Plan

The project plan is a brief document that describes the project objective, the risk analysis, schedule and resources. It is provided in Appendix A. Appendix B is WBS that indicates all the main required tasks. Appendix C is a PERT chart that shows task sequencing. Appendix D is a Gantt chart which lists the schedule for all tasks.

CHAPTER 5. SOFTWARE DEVELOPMENT APPROACH

This project is mainly a software development project. A sound software engineering approach is used to ensure high quality and timeliness of the software development. “High quality” means that the software meets its requirements. Software engineering is a young and rapidly maturing discipline which emphasizes the development of methodologies and tools to help manage the process of creating today’s complicated software. It is hard to define exactly. An early definition of software engineering was proposed by Fritz Bauer at the first major conference [29]:

The establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.

There are software engineering methodologies associated with such areas as software project planning, software requirements analysis, software design, software implementation and software testing. The use of a suitable software engineering approach enhances such software attributes as correctness, reusability, portability and reliability.

For PhyloSim a hybrid software development approach was used. The hybrid approach has the following elements:

(1) A baseline project plan was developed initially
(2) Project requirements were analyzed and a requirements specification was generated. The specification includes the model requirements and performance requirements. Best efforts were made to modify the software to increase the degree of performance.
(3) The software architecture and design were developed before implementation began. The architecture identified the major components, while the overall design includes the I/O, program structure and algorithm design.
(4) Individual phylogenetic models (discussed in chapter 3 and 7) were developed sequentially using an agile approach similar to extreme programming. Under this approach each phylogenetic model was developed and tested, followed by the next, etc. All such phylogenetic models used the same overall software framework or architecture.
(5) During the software development, software tools such as glade[30], Make utilities, and debuggers were used to help implement the project.
(6) Extensive testing of the software and all phylogenetic models was done using software verification and validation strategies. Written test plans were created. Test documentation was created which included the bug reports and comments on the results of the model computations.
(7) All software was well documented. The documentation includes source code with complete comments, structure and flowcharts of the program, project plan, requirement specification, test plan, and test documentation. These documents should make the software easy to read, maintain and change in the future.

Details of approach followed are provided in subsequent chapters. The software engineering approach helped to ensure project success.

In order to implement the software a hybrid model combining classical software engineering model and the Agile model with Extreme Programming approach was used. Each of the mathematical models were implemented and tested against other similar programs used for phylogenic inference. After the test showed satisfactory results the next model was implemented. This continued until all the models were implemented and then the entire system was tested.

CHAPTER 6. SOFTWARE REQUIREMENTS ANALYSIS

For complex systems, a complete understanding of software requirements is probably impossible to achieve at the beginning of the project. However, good understanding of the software requirements is usually the basis for a successful product. If we cannot understand the requirements of a particular software, then the output product will probably not satisfy the user and so the project can fail even if the project plan, the software design, implementation and testing are good. So, we need requirements analysis because it is the first step for ensuring that right product is developed. At this point, a requirements specification can be created. It describes the system, its main functions, its system environment, performance and constraints.

6.1 Technical requirements for the PhyloSim programmer

To implement this software the programmer needs to have a basic knowledge about genetics, computational biology, bioinformatics and especially mathematical modeling. This includes knowledge of how to parse Newick format trees, how to use glib data structure to get that information, how to store the tree data in appropriate data structures, and how to use them for computations. The use of multiple free libraries like gsl, glib, pango, atk, cairo is needed. The use of the glade GUI creator and corresponding gtk+ library can save a lot of time for implementation.

The development of software for Phylogenetic inference software is quite different from the development of other general software. It requires special skills and knowledge. I needed to learn how to design and write programs using gtk+, and other libraries to minimize execution time.

6.2 Input and Output requirements for PhyloSim

Input: The input data consists of a file with required parameters for each target model. Those parameters can be root distribution, relative rates, number of states, titv ratio, etc and a tree that describes the lineage and the branch length.

Output: The output data consists of the sequence of each of the leaves after the mutations have occurred down the tree according to the model parameters applied on the root sequence. These sequences can be DNA, amino acid, protein or custom made sequences. The program also supports having ancestral sequences; ancestral sequences are the sequences that are in the nodes of the tree.

6.3 Requirement specification

After analyzing the requirements, a formal requirements specification was done. The requirement specification is given as follows:

1. Introduction
   The document is the requirement specification for the software system for phylogenetic tree simulation based on complex and mixed models.

A GUI based software is to be developed for this purpose. This program will simulate the phylogenetic tree based on input model and its parameters. The program should be fast in execution and produce accurate results.

2. System Stakeholders

The main stakeholders are INBRE, advising committee members, and future PhyloSim users (expected to be).

The Committee

The committee consists of the developer and three committee members who are assisting the developer in making this product by giving computational biology, programming and software engineering advice.

Biotechnology Computing Research Group (BCRG)

The BCRG group at UAF's West Ridge is associated with the biotechnology and bioinformatics research at UAF. This software will add to the other software that the BCRG group has created and supports to the BioComputing research at UAF.

The BCRG existing portal could host the PhyloSim software, and make use of the parallel processing, that the Bioinformatics core already provides.

IdEA Network for Biomedical research excellence (INBRE)

INBRE is funding some of the bioinformatics research at UAF. PhyloSim development is funded by INBRE and supports the INBRE's goals.

Fairbanks Community

UAF is well known for its Biology research. PhyloSim helps UAF broaden and grow its research. It will help the ongoing research effort at UAF and also the biology and bioinformatics community in general.

Computational Biologist / Bioinformatics researchers

Phylogenetic Inference programs are widely used and needed by Computational Biologist and Bioinformatics researchers. New, user friendly, flexible and accurate phylogenetic inference software like PhyloSim helps this group in their pursuit of new ideas and assists in implementing them.

2.1 Functional Requirements

1. Simulation Function:

   Provides a way to generate a phylogenetic tree from the input Model parameters.

2. Multi Model or Model Mixer Function

   Allows the use of more than one model at different parts (nodes) of the same tree.

3. File formats

   Support popular file formats such as PHYLIP, NEXUS[31] and PAUP.

4. Model loader

   Supports the loading of different kinds of models.

5. Complex Model initiator

   Support options such as across site / lineage rate variation and other complex features.

6. Tree creator

   Allows the user to create trees and enter edge length and leaf names easily.

2.2 Non-Functional Requirements

2.2.1 Flexibility

The system must have flexibility in supporting variable input parameters. It can support variable matrix sizes which are considered very important for phylogenetic inference schemes. Other variable parameters are the relative-rates, custom bases and root parameters. The product also allows entering the matrix parameters directly in case of General Markov models.

2.2.2 Modifiability

PhyloSim must be able to implement new phylogenetic modules with minimal change costs. The way to achieve this requirement is to keep different mathematical models in different PhyloSim sub-modules; any of these modules can be changed to further enhance those models. Another desirable modification is to allow higher computational speeds with minimal software changes. Enhancement can be in the form of increasing computational speed such as creating a faster matrix-matrix multiplication for example.

2.2.3 Performance

PhyloSim must create phylogenetic trees using large DNA, codon, amino acid and protein bases quickly, for example, it should generate the output of a five taxon tree with a million bases in few seconds.

2.2.4 Usability

PhyloSim must be easy to use for its targeted primary users, who are computational biology and phylogenetics researchers. One way to achieve this is to support GUI, which enhances the capability of easier data entering and modifying. This viewing of input and output also supports changing of the different parameters.

2.2.5 Scalability

PhyloSim must install and run on a single computer system, such as a modern laptop. However, better performance could result by a further enhancement of the system by making it run on a parallel machine. This could be a future requirement for an advanced PhyloSim.

2.2.6 Extensibility

Many possible extensions could make PhyloSim more useful. Examples could be to add code to support newer models and a graphical representation of the output data.

2.2.7 Testability

Testability verifies the ease of testing the system. PhyloSim takes data and outputs files that can be used in other program to verify the accuracy of the program. Such programs are PAUP* and PhyML. They take the output file from this product and produce the same tree that was entered in our program. The Complexity Cluster service of BCRG, UAF was used to make use of PAUP* and check the GTR & GTR-like models for varying input. The web interface of PhyML was used for testing models as it is freely available online.

2.2.8 Compatibility

The system must be compatible with the different NEXUS, PHYLIP files available online for existing data of different species. The PhyloSim output file may be utilized by other phylogenetic inference software.

2.2.9 Reliability and Accuracy

Reliability and accurate results are major software design constraints.

2.2.10 Constraints on time and resources

PhyloSim must run on any Linux and MacOSX machine.
PhyloSim should be finished and ready for release by the end of July, 2008.

2.3 User Characteristics

User- A user is a person who can load input files, modify the existing parameters and make use of the output file. Targeted primary users are computational biology and phylogenetic researchers.

For benefit to some user who wants to modify PhyloSim themselves the entire source code will be made available under the GNU license so that they can change it according to their need.

7.1 Software Architecture

At this phase PhyloSim enters into the second stage of development. This part is the architecture part. The Architecture should lend itself to incremental implementation via the creation of a "skeletal" system in which the communication paths are exercised but which at first has minimal functionality. This skeletal system can then be used to "grow" the system incrementally, easing the integration and testing efforts[27]

Also, PhyloSim software architecture should be such as to faciliate achievement of the non-functional requirements(flexibility, modifiability) that were discussed in chapter 6.

Software architecture is much concerned with software structure, i.e. the identification of the main software componenet and their inter-relationships.

The divide and conquer technique is used to make the project easy to implement, it consists of the identification of the PhyloSim project tasks. These are shown below in several categories.

In its simplest form, the PhyloSim Software system consists of

1. Parser: This is the one of core Module of the system, it takes input data and parses it and feeds the models the required input.
2. GUI Module: This Module helps user create input file using template reads in model data from input text file and views or saves output as user desires.
3. Mathematical Model loader: This module loads any mathematical model according to the description of the Input txt file. It simulates the phylogenetic tree and outputs the DNA sequences in different branches and produces the resultant leaf nodes with DNA sequences.

The system was chosen to be built, because it is a somewhat unique system that mixes up new and complex models that other available systems do not support. The system helps biologists to create new trees that are supposed to be more biologically realistic.

PhyloSim has major software components (modules) which are identification in the table below.

The Program modules are:

Module name	Functions
PhyloSim.c	Program entry point. GUI load and initialization.
interface.c	GUI management.
interface.glade	XML description of GUI elements, for using with libglade.
parser.c	Input data parsing.
data.c	Common-used model descriptions and matrix operations.

7.2 Software Design

After Architecture part starts the design part. Software design is a process through which requirements are translated into a representation of software[29]. Good design involves making the simplest system that will satisfy the requirements of the project[32].

The description of some of the these modules is provided below

7.2.1 Design of PhyloSim.c

This module contains program entry point function main(). It initializes GTK+2, creates and maximizes main window and enters GTK+2 event loop. After initialization, program execution is controlled by **interface.c** module.

7.2.2. Design of interface.c

This module allows user to control the program execution. Using corresponding GUI controls, user can start calculations on currently loaded text data.
We can see the algorithm of the execution function in Figure 1 in next page.

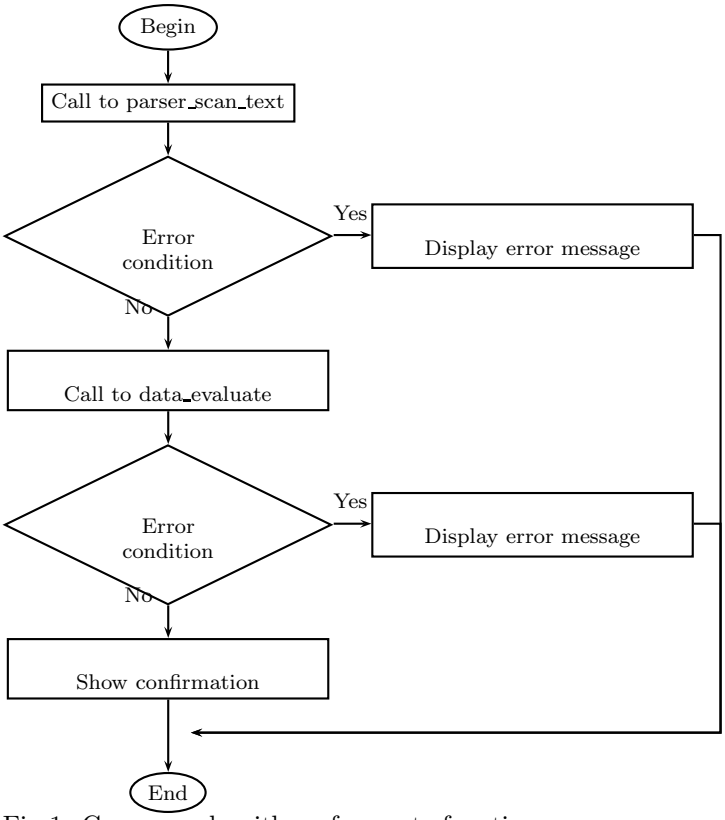


Fig 1: Common algorithm of execute function:

7.2.3. Design of parser.c

This module contains lexical scanning and parsing related functions. Parsing process of entire input text is controlled by function **int parser_scan_text (const char * text, parser_data_t * data)**. It takes input data text as **const char * text** and scanning over it using **GScanner** lexical scanner included in *glib* software library.

The **parser_data_t** structure contains parsed parameters as dynamically allocated variables of corresponding type. Before scanner find a token corresponding to particular data entry in **parser_data_t** structure, the structure entry keeps **NULL** value. After finding a token, parser dynamically allocates memory area, fills it with value read from input and stores pointer to the allocated memory in **parser_data_t** structure. If corresponding slot is already filled with some data, parser generates error state due duplicated data entries is not allowed.

Common algorithm of parsing process is shown in Figure 2 as flowchart.

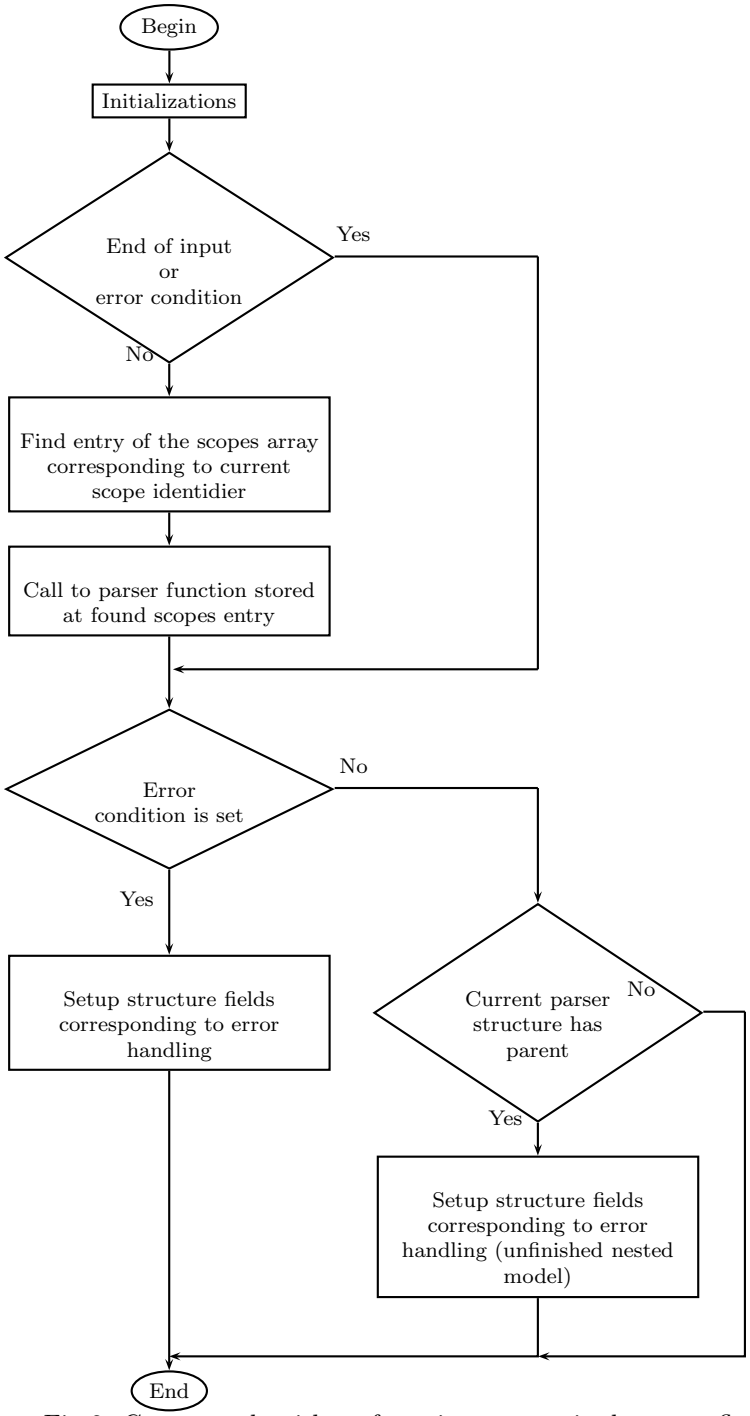


Fig 2. Common algorithm of parsing process is shown on flowchart.

7.2.4 Design of data.c

This module controls data integrity checking, calculations execution and contains auxiliary mathematical functions. Internally, it contains static structures describing models and states sets. **const data_model_t data_models []** array contains model information, such as internal identifier, model name, calculations function and Q matrix constructing function. Last one can be NULL if such model cannot make Q matrix.

Static array of anonymous structures **data_parser_items** contains enumerated set of data fields and corresponding names. It used in **int data_check_valid_parser_set (parser_data_t * p, unsigned long int set, unsigned long int mayset, int ms, char ** msg)** function which checks that given parser data contains only desired fields but not other ones.

The data flowchart is shown in next page as Figure 3:

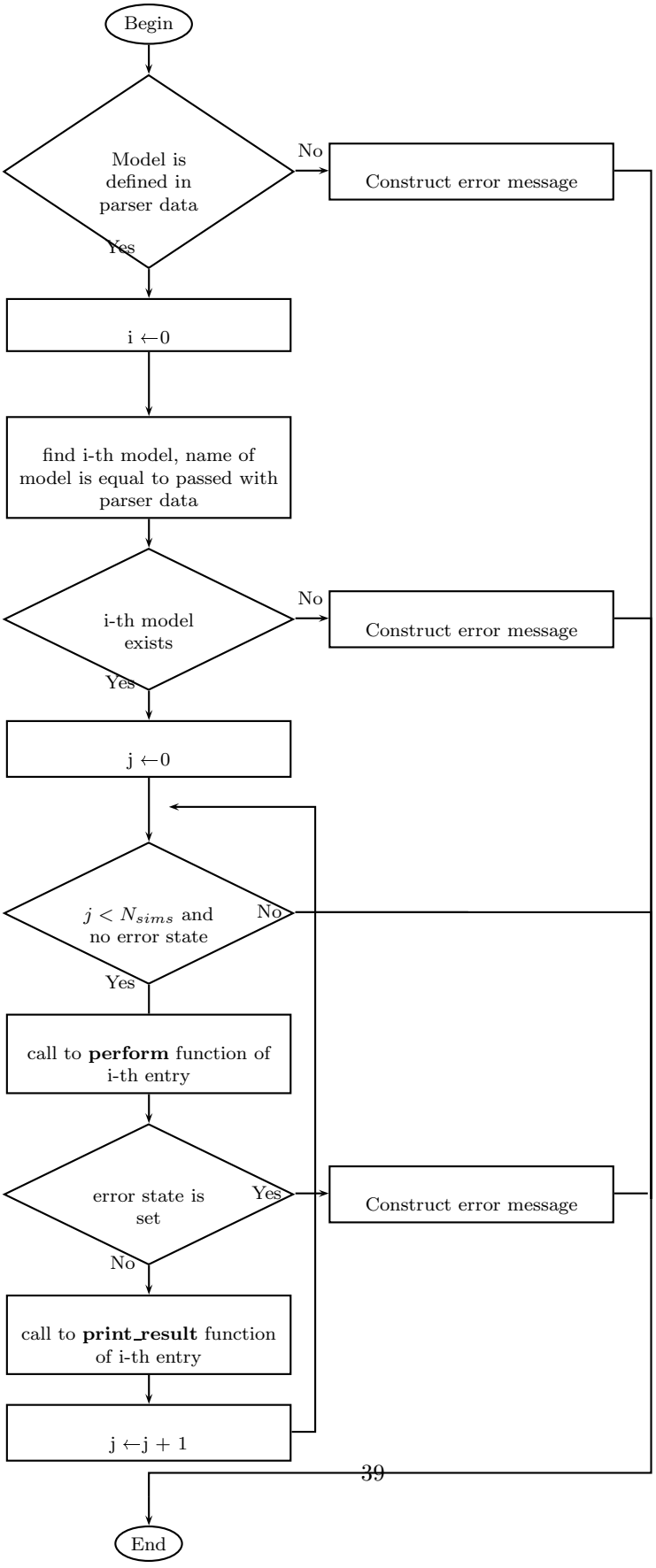

Begin
Model is defined in parser data
No
Construct error message
Yes
i ←0
find i-th model, name of model is equal to passed with parser data
i-th model exists
No
Construct error message
Yes
j ←0
$j < N_{sims}$ and no error state
No
Yes
call to **perform** function of i-th entry
error state is set
Yes
Construct error message
No
call to **print_result** function of i-th entry
j ←j + 1
End

Fig 3. Common algorithm of data.c process is shown in flowchart

Next page in Figure 4. shows the overall structure of the PhyloSim program

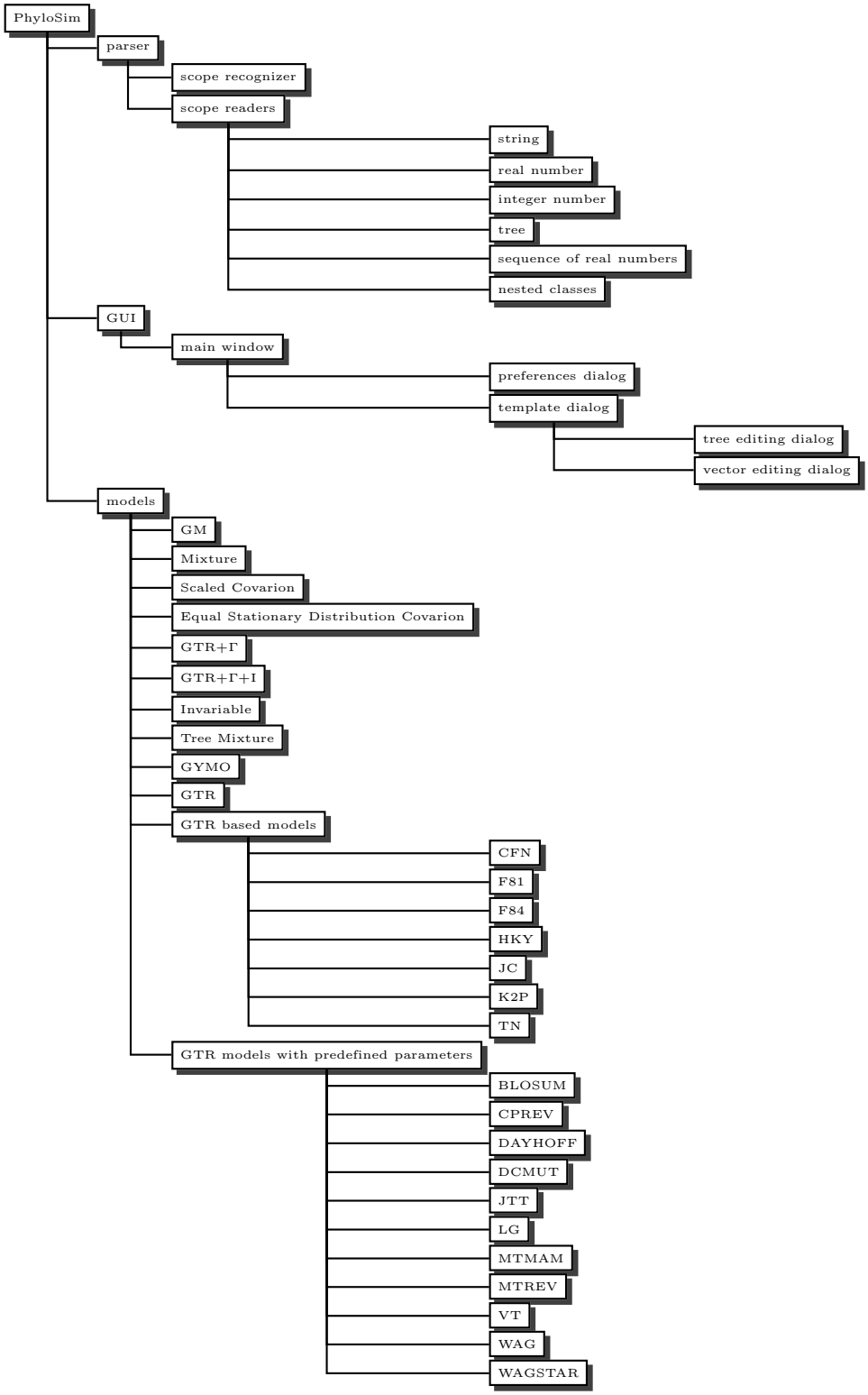


Fig 4. Overall structure of the Software Architecture of PhyloSim

7.3 Software Implementation

Currently PhyloSim supports the following phylogenetic models:

1. GM
2. Mixture
3. Scaled Covarion
4. Equal Stationary Distribution Covarion
5. GTR+Γ
6. GTR+Γ+I
7. Invariable
8. Tree Mixture
9. GYMO
10. GTR
11. GTR based models
    (a) CFN
    (b) F81
    (c) F84
    (d) HKY
    (e) JC
    (f) K2P
    (g) TN
    (h) GTR models with predefined parameters (BLOSUM, CPREV, DAYHOFF, DCMUT, JTT, LG, MTMAM, MTREV, VT, WAG, WAGSTAR)

Code implementing models is placed on *models* subdirectory. There is a pair of C source code file and it's header file for each model named **do_*xxx*.c** and **do_*xxx*.h**, where ***xxx*** is signature related to model name. For each model module exports function **int do_*xxx* (parser_data_t * data, void (* print_fn) (void *, const char *, ...), void * print_data, char ** errmsg)** which performs calculation for model. Some models also exports function **int *xxx*_make_q (parser_data_t * data, long int ms, double ** ret, double * root)** which calculates Q matrix for that model. These functions are enlisted in states array **const data_model_t data_models []** of *data.c* module with model name used in the input data. Function **int do_*xxx* (parser_data_t * data, void (* print_fn) (void *, const char *, ...), void * print_data, char**

**** errmsg)** uses **parser_data_t * data** as input data, prints out calculation result using passed **print_fn (print_data, *format*, ...**. On error each function returns nonzero value and set *** errmsg** to allocated string corresponding to error found.

7.3 GM model, do_gm.c module

This model uses the following calculations algorithm:

1. call to **prepare** function which checks input data and constructs **gm_data_t** model specific data structure. That structure contains plain arrays instead of *glib* specific **GList** and **GHashTable** data structures which is hard to use in calculation functions.
2. call to **init** function which initializes random numbers generator and creates root DNA sequence.
3. call to *glib*'s **g_node_children_foreach** function which calls **do_mutation** function for each child node of root of the input tree.

do_mutation function:

1. calls **data_mutator** function, passing to it **bounds** vector and DNA sequence of current node, DNA sequence of parent node.
2. prints DNA sequence of current node.
3. call to *glib*'s **g_node_children_foreach** function which calls **do_mutation** function for each child node of current one.

7.3.2. Mixture model, do_mix.c module

This model uses the following calculation algorithm:

1. Check input data
2. Allocate memory and initialize variables.
3. For each of nested models construct $\tilde{Q}_i$ matrix using **data_make_q_matrix_for_model**.
4. Construct Q matrix:

$$Q = \begin{bmatrix} \tilde{Q}_1 & 0 & 0 & \dots \\ 0 & \tilde{Q}_2 & 0 & \dots \\ 0 & 0 & \tilde{Q}_3 & \dots \\ \dots & \dots & \dots & \tilde{Q}_n \end{bmatrix}$$

5. Construct μ vector using class root distribution π and root elements of nested models:

$$\mu = [\pi_1\alpha_{1,1}, \pi_1\alpha_{1,2}, \dots, \pi_2\alpha_{2,1}, \pi_2\alpha_{2,2}, \dots]$$

where $\alpha_{i,j}$ is j-th root element of i-th model.

6. Construct T, T^{-1} matrices and λ vector using **data_make_t_from_q_matrices** function of *data.c* module.
7. Initialize model specific structure **mix_data_t**.
8. Construct amino acid states array.
9. Construct model specific tree by calling **make_mix_node** function for each node of source tree.
10. Create root DNA sequence.
11. Use *glib* function **g_node_children_foreach** call **do_mutation** function for each node of tree root.
12. Free allocated memory.

make_mix_node function does next:

1. Allocate memory for node.
2. Construct edge matrix for node using **data_make_edge** function of *data.c* module.
3. Construct bounds matrix using **data_make_bounds** function of *data.c* module.

do_mutation function is standard function constructed for this model by **DATA_MAKE_GTR_LIKE_MUTATION_FN** macro declared in *data.h*.

7.3.3 Scaled Covarion, do_cov.c module

This model uses the following calculation algorithm:

1. Check input values.
2. Allocate memory.
3. From state relative rates and state root distribution construct $\tilde{Q}$ matrix using **data_make_q_matrix** function of *data.c* module.
4. From class relative rates and class root distribution construct $\tilde{S}$ matrix using **data_make_q_matrix** function of *data.c* module.
5. From class root distribution σ and state root distribution π construct μ vector:
$$\mu_{Ni+j} = \sigma_i * \pi_j$$
where N is number of states.
6. Construct Q matrix:
$$Q = \begin{bmatrix} r_1\tilde{Q} + \tilde{S}_{1,1}I & S_{1,2}I & S_{1,3}I & \dots \\ S_{2,1}I & r_2\tilde{Q} + \tilde{S}_{2,2}I & S_{2,3}I & \dots \\ S_{3,1}I & S_{3,2}I & r_3\tilde{Q} + \tilde{S}_{3,3}I & \dots \\ \dots & \dots & \dots & r_n\tilde{Q} + \tilde{S}_{n,n} \end{bmatrix}$$

7. Calculate $\nu = -\sum_i Q_{i,i}\mu_i$
8. Rescale Q matrix with ν:

$$Q_{i,j} \leftarrow Q_{i,j}/\nu$$

9. Initialize model specific structure **cov_data_t**.
10. Construct states array.
11. Construct T, T^{-1} matrices and eigenvalues vector using **data_make_t_from_q_matrices** function of *data.c* module.
12. Construct model specific tree by traversing source tree using **g_node_copy_deep** function of *glib* and calling **make_cov_node** function for each source tree node.
13. Create root DNA sequence
14. Call to *glib*'s **g_node_children_foreach** function which calls **do_mutation** function for each child node of root of the input tree.

make_cov_node function does next:

1. Allocate memory for node.
2. Construct edge matrix for node using **data_make_edge** function of *data.c* module.
3. Construct bounds matrix using **data_make_bounds** function of *data.c* module.

do_mutation function is standard function constructed for this model by **DATA_MAKE_GTR_LIKE_MUTATION_FN** macro declared in *data.h*.

7.3.4. Equal Stationary Distribution Covarion, do_ecov.c module

This model uses the following calculation algorithm:

1. Check input values.
2. Allocate memory.
3. From class root distribution σ and state root distribution π construct μ vector:

$$\mu_{Ni+j} = \sigma_i * \pi_j$$

where N is number of states.

4. From class relative rates and class root distribution construct $\tilde{S}$ matrix using **data_make_q_matrix** function of *data.c* module.

5. Construct Q matrix:

$$Q = \begin{bmatrix} \tilde{Q}_1 + \tilde{S}_{1,1}I & S_{1,2}I & S_{1,3}I & \dots \\ S_{2,1}I & \tilde{Q}_2 + \tilde{S}_{2,2}I & S_{2,3}I & \dots \\ S_{3,1}I & S_{3,2}I & \tilde{Q}_3 + \tilde{S}_{3,3}I & \dots \\ \dots & \dots & \dots & \tilde{Q}_n + \tilde{S}_{n,n} \end{bmatrix}$$

   where $\tilde{Q}_i$ constructed from state root distribution π and relative rates of i-th nested class using **data_make_q_matrix** function of *data.c* module.

6. Calculate $\nu = -\sum_i Q_{i,i}\mu_i$

7. Rescale Q matrix with ν:

$$Q_{i,j} \leftarrow Q_{i,j}/\nu$$

8. Initialize model specific structure **ecov_data_t**.

9. Construct states array.

10. Construct T, T^{-1} matrices and eigenvalues vector using **data_make_t_from_q_matrices** function of *data.c* module.

11. Construct model specific tree by traversing sourse tree using **g_node_copy_deep** function of *glib* and calling **make_ecov_node** function for each sourse tree node.

12. Create root DNA sequence

13. Call to *glib*'s **g_node_children_foreach** function which calls **do_mutation** function for each child node of root of the input tree.

make_ecov_node function does next:

1. Allocate memory for node.

2. Construct edge matrix for node using **data_make_edge** function of *data.c* module.

3. Construct bounds matrix using **data_make_bounds** function of *data.c* module.

do_mutation function is standard function constructed for this model by **DATA_MAKE_GTR_LIKE_MUTATION_FN** macro declared in *data.h*.

7.3.5. GTR+Γ,do_gtr+r.c module

This model uses the following calculation algorithm:

1. Check input values.

2. Allocate memory.

3. Construct Q matrix using **gtr_r_make_q_checked**:

$$Q = \begin{bmatrix} r_1\tilde{Q} & 0 & 0 & \dots \\ 0 & r_2\tilde{Q} & 0 & \dots \\ 0 & 0 & r_3\tilde{Q} & \dots \\ \dots\dots\dots\dots & & & r_n\tilde{Q} \end{bmatrix}$$

where r_i calculated using **data_get_r** function of *data.h* module, and $\tilde{Q}_i$ constructed from root distribution π and relative rates using **data_make_q_matrix** function of *data.c* module.

Same function returns $\vec{\mu} = \{\underbrace{\frac{\vec{\pi}}{k}, \dots}_{k}\}$, where k is number of classes.

4. Initialize model specific structure **gtr_r_data_t**.
5. Construct states array.
6. Construct T, T^{-1} matrices and eigenvalues vector using **data_make_t_from_q_matrices** function of *data.c* module.
7. Construct model specific tree by traversing source tree using **g_node_copy_deep** function of *glib* and calling **make_gtr_r_node** function for each source tree node.
8. Create root DNA sequence
9. Call to *glib*'s **g_node_children_foreach** function which calls **do_mutation** function for each child node of root of the input tree.

make_gtr_r_node function does next:

1. Allocate memory for node.
2. Construct edge matrix for node using **data_make_edge** function of *data.c* module.
3. Construct bounds matrix using **data_make_bounds** function of *data.c* module.

do_mutation function is standard function constructed for this model by **DATA_MAKE_GTR_LIKE_MUTATION_FN** macro declared in *data.h*.

7.3.6. GTR+Γ+I,do_gtr+r+i.c module

This model uses the following calculation algorithm:

1. Check input values.
2. Allocate memory.

3. Construct Q matrix using **gtr_r_i_make_q_checked**:

$$Q = \begin{bmatrix} r_1\tilde{Q} & 0 & 0 & 0 & \dots \\ 0 & r_2\tilde{Q} & 0 & 0 & \dots \\ 0 & 0 & r_3\tilde{Q} & 0 & \dots \\ \dots & \dots & \dots & r_n\tilde{Q} & 0 \\ \dots & \dots & \dots & \dots & Z \end{bmatrix}$$

where Z - zero matrix, r_i calculated using **data_get_r** function of *data.h* module, and $\tilde{Q}_i$ constructed from root distribution π and relative rates using **data_make_q_matrix** function of *data.c* module.

Same function returns $\vec{\mu} = \{\underbrace{\vec{\pi}\frac{q}{k}, \dots}_{k}, P_{inv}\vec{\pi}\}$, where k is number of classes, $q = 1 - P_{inv}$.

4. Initialize model specific structure **gtr_r_i_data_t**.
5. Construct states array.
6. Construct T, T^{-1} matrices and eigenvalues vector using **data_make_t_from_q_matrices** function of *data.c* module.
7. Construct model specific tree by traversing source tree using **g_node_copy_deep** function of *glib* and calling **make_gtr_r_i_node** function for each source tree node.
8. Create root DNA sequence
9. Call to *glib*'s **g_node_children_foreach** function which calls **do_mutation** function for each child node of root of the input tree.

make_gtr_r_i_node function does next:

1. Allocate memory for node.
2. Construct edge matrix for node using **data_make_edge** function of *data.c* module.
3. Construct bounds matrix using **data_make_bounds** function of *data.c* module.

do_mutation function is standard function constructed for this model by **DATA_MAKE_GTR_LIKE_MUTATION_FN** macro declared in *data.h*.

7.3.7. Invariable, do_inv.c module

This model constructs root DNA sequence using root distribution parameter and copy it unchanged to all of input tree taxons.

7.3.8. Tree Mixture, do_tmix.c module

This model after validating input data calls all of nested models calculations. Next, it collects resultant sequences for each taxon depending on mode parameter passed in the input data set.

Random mode

Each i-th character of resultant sequence for a taxon picked up randomly from character with same index in sequence of same taxon of one of nested models with probability of selecting that model is equal to class root distribution element corresponding to that model.

Concatenated mode

Each class root distribution element multiplified to lognormal distributed random number. New randomized class root distribution is rescaled so it keeps to sums to 1. Resultant sequence is concatenated form subsequences of nested models. Length of each submodel sequence is proportional to randomized class root distribution element corresponding to that submodel.

7.3.9. GYMO, do_gymo.c module

This model using 4 values of root parameters as $\pi_A, \pi_C, \pi_G, \pi_T$. After validating input data, this model constructs μ vector with length of 61: $\mu_i = \pi_{idx(i,1)}\pi_{idx(i,2)}\pi_{idx(i,3)}$. Indices *idx(i, j)* is calculated from list of all codons AAA, AAC, AAG, AAT, ACA, ACC, ... with TAG, TAA and TGA codons omited. So, $\mu_1 = \pi_A^3$, $\mu_2 = \pi_A^2\pi_C$, etc. Values of μ vector is rescaled so it sums to 1.0. Next, Q matrix costructed as

$$Q_{i,j} = \begin{cases} 0, & \text{if } codon_i \text{ and } codon_j \text{ differs in 2 or more locations} \\ \mu_i, & \text{if } codon_i \text{ and } codon_j \text{ representing the same amino acid and change is transversion} \\ \mu_i k, & \text{if } codon_i \text{ and } codon_j \text{ representing the same amino acid and change is transition} \\ \mu_i \omega, & \text{if } codon_i \text{ and } codon_j \text{ representing different amino acids and change is transversion} \\ \mu_i \omega k, & \text{if } codon_i \text{ and } codon_j \text{ representing different amino acid and change is transition} \end{cases}$$

Where ω is value passed in model by *nonsynonomous ratio* keyword and k is value passed in model by *titv ratio* keyword. Q matrix is used for constructing T, T^{-1} matrices and eigenvalues vector using **data_make_t_from_q_matrices** function of *data.c* module. These matrices is used for constructing edge matrices for taxons, just like in any other GTR based model.

States list for this model is array of integers in range from 1 to 61. Output values could be printed out as codons or as amino acids. Output style is managed by *output* keyword which can take values *DNA* or *protein*.

7.3.10. GTR, do_gtr.c module

This model uses next calculation algorithm:

1. Initialize model specific structure **gtr_data_t**.
2. Call to **prepare** function which checks input data and fill model specific structure **gtr_data_t** with data get from input **parser_data_t**.
3. Call to **init** function which creates root DNA sequence.
4. call to *glib*'s **g_node_children_foreach** function which calls **do_mutation** function for each child node of root of the input tree.

do_mutation function is standard function constructed for this model by **DATA_MAKE_GTR_LIKE_MUTATION_FN** macro declared in *data.h*.

7.3.11. GTR based models

All of these models have similar structure of code. For each model is defined function **make_root**. Functions **check_valid_data**, ***xxx*_make_q** and **do_*xxx*** for each model is constructed by corresponding macros **DATA_MAKE_CHECK_VALID_DATA_FN**, **DATA_MAKE_Q_FN** and **DATA_MAKE_DO_FN** defined in *data.c* module. For performing actual calculations, all of these models is calling to **do_gtr_predefined** which take model specific values of root distribution and relative rates as input parameter instead of extracting them from parser data.

7.3.11a. CFN, do_cfn.c module

Used for states length of 2. Relative rates is [1], root distribution is [0.5, 0.5].

7.3.11b. F81, do_f81.c module

Used for states length of 4. Relative rates is [1, 1, 1, 1, 1, 1], root distribution accepted from input data.

7.3.11c. F84, do_f84.c module

Used for states length of 4. Root distribution accepted from input data. Relative rates is $[1, 1 + \frac{J}{\pi_R}, 1, 1, 1 + \frac{J}{\pi_Y}, 1]$, where R - input value of titvratio,

$$\pi_R = \pi_1 + \pi_3$$

$$\pi_Y = \pi_2 + \pi_4$$

$$J = \frac{R\pi_R\pi_Y - \pi_1\pi_3 - \pi_2\pi_4}{\frac{\pi_1\pi_3}{\pi_R} - \frac{\pi_2\pi_4}{\pi_Y}}$$

7.3.11d. HKY, do_hky.c module

Used for states length of 4. Root distribution accepted from input data. Relative rates is $[1, K, 1, 1, K, 1]$, where R - input value of titvratio,

$$\pi_R = \pi_1 + \pi_3$$

$$\pi_Y = \pi_2 + \pi_4$$

$$K = \frac{R\pi_R\pi_Y}{\pi_1\pi_3 + \pi_2\pi_4}$$

7.3.11e. JC, do_jc.c module

Used for states length of 4. Relative rates is [1, 1, 1, 1, 1, 1], root distribution is [0.25, 0.25, 0.25, 0.25].

7.3.11f. K2P, do_k2p.c module

Used for states length of 4. Relative rates is $[1, 2a, 1, 1, 2a, 1]$, where a - input value of titvratio, Root distribution is [0.25, 0.25, 0.25, 0.25].

7.3.11g. TN, do_tn.c module

Used for states length of 4. Relative rates is $[1, 1, 1, 1, 1, 1]$, where a - input value of titvratio, Root distribution is [0.25, 0.25, 0.25, 0.25].

7.3.11h. BLOSUM, CPREV, DAYHOFF, DCMUT, JTT, LG, MTMAM, MTREV, VT, WAG, WAGSTAR

Used for states length of 20. Each of this models is GTR model with fixed root states and relative rates parameters set.

CHAPTER 8. SOFTWARE TEST, TEST RESULTS AND DISCUSSION

Planning the project carefully, writing a requirement specification and making a good design are the initial steps to produce a high quality product. Another quality assurance technique used during this project is testing. The aim of the testing is to show that the system does what it is supposed to do. Testing involves checking whether the system satisfies the entire requirement and does all the arithmetic and logic correctly. Often the number of conditions that should be checked becomes quite large. So a test plan was created to check as many conditions as possible. A test plan is also dynamic. When errors are found, they should be removed. This elimination of errors can cause kinds of errors. So the test plan needs to be adaptable.

8.1 Test plan

Enough time and testing effort should be spent to ensure PhyloSim quality. The plan includes functions to be tested and procedure and approaches to be used. However, for complex software with extensive data calculations it is impossible to test every detail of the software. To test the software effectively and efficiently, a test plan with well designed test cases was used.

8.2 Test case

Test case design is an important step during software test planning. The test cases should cover the program comprehensively and identify any weaknesses.

8.2.1 Test of the System

The quality of the Mathematical Model Module is essential to the quality of the overall program. To ensure that we have used the following two type of testing.

8.2.2 White Box Testing

Basis path testing, control structure testing and loop testing were used. The main logical conditions and paths contained in the software were tested using the debugger.

8.2.3 Black Box Testing

Black box testing [29] is a method to test the general functions of the software. In this project all the implemented models were tested.

Models tested were,

1. GM
2. Mixture
3. Scaled Covarion
4. Equal Stationary Distribution Covarion
5. GTR+Γ
6. GTR+Γ+I
7. Invariable

8. Tree Mixture
9. GYMO
10. GTR
11. GTR based models
    11.1. CFN
    11.2. F81
    11.3. F84
    11.4. HKY
    11.5. JC
    11.6. K2P
    11.7. TN
    11.8. GTR models with predefined parameters (BLOSUM, CPREV, DAYHOFF, DCMUT, JTT, LG, MTMAM, MTREV, VT, WAG, WAGSTAR)

All of these models (except model 2, 3 & 8) were tested with popular phylogenetic software PAUP* and PhyML. The output of our program was fed into these program and they inferred the tree back. Model 2, 3 & 8 were tested against output of MATLAB code that was written for testing these models.

8.3 Performance Test

Execution time was fast in comparable with other phylogenetic inference software for producing output sequence for the models.

8.4 Test documentation

Test documentation is the report of the testing procedure. It indicates what has been done and what still needs to be done. An example of the Test documentation is given in appendix E.

CHAPTER 9: SUMMARY AND CONCLUSIONS

In this project the main tasks accomplished were the following:

- Software was designed for high quality and high performance
- A new Phylogenetic Software system called PhyloSim was created, implementing many models and improved structure, with more added features than other existing inference software.
- New input commands were created to enter the description of new models in the system
- Model templates were developed and used to enter data seamlessly
- A GUI was developed for the phylogenetic tree editor
- The developed Software was tested with good results, including

  Verification: The results of the program were same as those of Seq-Gen and PhyML for the models that they support.

  Validation: The input data and the tree were in excellent agreement with the output data (they had the approximate same probability).

  Speed: For most of the models execution time was satisfactory.

All of the modules developed were useful, and PhyloSim is a good simulation software system for phylogeny inference. All the functional and non-functional requirements were satisfied. Some snapshots of the program are given in Appendix G.

The PhyloSim modeling process replicates the inputs and products of the original modeling process but optimizes the modeling process with computer automation in mind. It provides a blueprint for building new modeling tools that advance the state of the art in modeling.

A study of some well known phylogenetic tree software has revealed that its documentation is inadequate. Programming teams that build simulation tools should better document their corresponding simulation processes. Because such documentation allows for analysis of the design of the tool without confusing with the implementation of the tool. Moreover, computational biology is a still-expanding field, and mathematical model tool builders can expect that researchers will demand support for newer phylogenetic models and more modeling activities. If simulation tool builders do not record their modeling process, the adaptation of such tools to new activities is difficult. For PhyloSim we have created enough documentation so that programmers can tailor it to new needs and use the existing PhyloSim modules to create newer models.

PhyloSim provides a flexible and efficient source of simulation for a wide range of substitution models, it is a useful tool for researchers studying phylogenetic relationships, molecular evolution of biological sequences, and supports the ability to infer relationship from data correctly.

During the PhyloSim development project, suitable software engineering techniques and project management methods were used. This approach helped the project be successfully implemented. Because

of the good project plan, requirements analysis and design, a structured program was developed. The structure of the source code is less coupled and highly cohesive, so the maintainability of the program is excellent. Extensive testing of the PhyloSim verified its high reliability.

When complex software is developed, a software engineering approach is usually helpful regardless of the application area. So software engineering is not only for software engineers in computer science but also for computational biology researchers who can use it in their world. In general project planning, requirements analysis and design methodologies avoid much trouble and save cost and time in implementation [33].

CHAPTER 10. RECOMMENDATION FOR FUTURE WORK

The computational biology community is in sore need of tools for efficiently, reliably, and repeatedly building large mathematical models. PhyloSim supports larger, more complex models than comparable modeling tools. It is especially suitable for creating newer phylogenetic models. Many mathematical models were successfully implemented and shown to produce correct output. Their execution time was also quite small.

If higher execution speeds are required then the following possibilities should be considered

1. Try using the explicit shared memory method or the message passing method or other newer methods for parallel programming

   By using parallel programming mutation can be made to happen in different branches at the same time. Various parallel programming methods might produce better performance.

2. Create new mathematical models

   Newer mathematical models that produce more biologically realistic outputs more quickly. If simpler equations can be found to perform the same functions, then better performance might be possible.

3. Implement better algorithms for the software system

   Newer faster algorithms for PhyloSim might be used which when coded execute faster and minimize output time. (see chapter 7 for examples of algorithms used in PhyloSim)

Another way to improve the PhyloSim might be to make it web based. With such a system, users can easily access it by uploading their input file and either downloading the output shown or having the link emailed to them. However, this might requires bandwidth for file transfers.

PhyloSim was developed as a standalone system serving a single user. However, its further use could be encouraged by making a version that would support multiple users simultaneously.

GLOSSARY

Bioinformatics: refers to the creation and advancement of algorithms, computational and statistical techniques, and theory to solve formal and practical problems arising from the management and analysis of biological data.

BCRG: Biotechnology Computing Research Group. The Biotechnology Computing Research Group at UAF provides programming and High Performance Computing (HPC) support to the Life Sciences Community, and encourages interdisciplinary education within Biology, Computer Science, Mathematics and Engineering.

cygwin: Cygwin is a Linux-like environment for Windows.

Covarion: The method of covarions, or concomitantly variable codons, is a technique in computational phylogenetics that allows the hypothesized rate of molecular evolution at individual codons in a set of nucleotide sequences to vary in an autocorrelated manner.

DNA: Deoxyribonucleic acid (DNA) is a nucleic acid that contains the genetic instructions used in the development and functioning of all known living organisms.

GTR+Γ, GTR+Γ+I, F84, HKY (K2P, F81 and JC69), JTT, WAG, PAM, BLOSUM, MTREV and CPREV: Widely used mathematical models for inferring phylogeneic relationships among species.

gsl: GNU Scientific Library is a numerical library for C and C++ programmers

Gantt Chart: A chart which lists the schedule, resources allocation, critical path and other information for the project.

Glade: Glade - a User Interface Designer for GTK+ and GNOME. Glade is a RAD tool to enable quick & easy development of user interfaces for the GTK+ toolkit and the GNOME desktop environment, released under the GNU GPL License.

Inferring Relationship: Finding relationships among species.

INBRE: IDeA Network for Biomedical Research Excellence.

Mathematical model: A mathematical model is an abstract model that uses mathematical language to describe the behavior of a system.

Phylogeny: The history of species lineages as they change through time.

Phylogenetic tree: also called an evolutionary tree, which shows the evolutionary relationships among various biological species or other entities that are believed to have a common ancestor.

PAUP*: Phylogenetic Analysis Using Parsimony has made it the most widely used software package for the inference of evolutionary trees.

PERT Chart (Performance Evaluation Review Techniques): A network which lists the sequence of the main tasks in the project.

Seq-Gen: Seq-Gen is a program that will simulate the evolution of nucleotide or amino acid sequences along a phylogeny, using common models of the substitution

WBS (Work Breakdown Structure): A chart which lists all main tasks to be done in the project.

REFERENCES

[1] Elizabeth S. Allman and John A. Rhodes. The identifiability of tree topology for phylogenetic models, including covarion and mixture models. J. Comput. Biol. 13:1101-1113, 2006.

[2] Mark Pagel and Andrew Meade. A phylogenetic Mixture Model for Detecting Pattern-Heterogeneity in Gene Sequence or Character-State Data. Syst Biolo. 53:571-581, 2004.

[3] Daniel Stefankovic and Eric Vigoda. Phylogeny of Mixture Models: Robustness of Maximum Likelihood and Non-identifiable Distributions. J. Comput.Biol. 14:156-189, 2007.

[4] Cécile Ané and Gordon Burleigh and Michelle M. McMahon and Micheal J. Sanderson. Covarion Structure in Plastid Genome Evolution: A New Statistical Test. Mol. Biol. Evol., 22:914-924, 2005.

[5] Nicolas Galtier. Maximum-Likelihood Phylogenetic Analysis under a covarion-like Model. Mol. Biol. Evol., 18:866-873, 2001.

[6] David Penny and Bennet J. McComish and Micheal A. Charleston and Michael D. Hendy. Mathematical Elegance with Biochemical Realism: The Covarion Model or Molecular Evolution. J. Mol. Evol., 53:711-723, 2001.

[7] Tuffley, Chris and Steel, Mike. Modeling the covarion hypothesis of nucleotide substitution. Math. Biosci. 147:63-91,1998.

[8] Joseph Felsenstein. Phylip: Phylogenetic inference package. version 3.6. University of Washington. 2004.

[9] F. Ronquist and J. P. Huelsenbeck. MRBAYES: Bayesian inference of phylogenetic trees. Bioinformatics. 7(8):754-5, 2001.

[10] F. Ronquist and J. P. Huelsenbeck. Bayesian phylogenetic inference under mixed models. Bioinformatics. 19:1572-1574, 2003.

[11] D. Swofford. PAUP: Phylogenetic analysis using parsimony (* and other methods). Version 4.0. Sinauer Associates, 2002.

[12] Seq-Gen http://tree.bio.ed.ac.uk/software/seqgen/

[13] Andrew Rambaut and Nicholas C. Grassly. Seq-Gen: an application for Monte Carlo simulation of DNA sequence evolution along phylogenetic trees. 13:235-238, 1997.

[14] Vivak Jayaswal and John Robinson and Lars Jermiin. Estimation of Phylogeny and Invariant Sites under the General Markov Model of Nucleotide Sequence Evolution. Syst Biol. 56:155-162, 2007.

[15] Elizabeth S. Allman and John A. Rhodes. Mathematical Models in Biology, An Introduction., Cambridge University Press, 2004.

[16] Joseph Felsenstein. Inferring Phylogeneies. Sinaer Associates, 2004.

[17] Edelstein-Keshet , Mathematical Models in Biology, SIAM,1988.

[18] Huelsenbeck, John P. Performance of Phylogenetic methods in Simulation. Sys. Biol., 44:17-48, 1995.

[19] Huelsenbeck, John P. and David M. Hillis. Success of phylogenetic methods in the four-taxon case. Sys Biol. 42:247-264, 1993

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

[21] PAUP* http://paup.csit.fsu.edu/

[22] John P Huelsenbeck and Marc A Suchard. A nonparametric method for accommodating and testing across-site rate variation. Syst Biol. 56 (6):975-87, 2007

[23] GTK+ - The GIMP Toolkit, http://www.gtk.org/

[24] James Archie, William H.E. Day, Wayne Maddison, Christopher Meacham, F. James Rohlf, David Swofford, and Joseph Felsenstein, Newick Tree format http://evolution.genetics.washington.edu/phylip/newicktree.html

[25] Daniel H. Huson, Dendroscope - An interactive viewer for large phylogenetic trees, http://www-ab.informatik.uni-tuebingen.de/software/dendroscope/welcome.html

[26] GSL - GNU Scientific Library http://www.gnu.org/software/gsl/

[27] Len Bass, Paul Clements, Rick Kazman, Software Architecture in Practice, Second Edition, 2004.

[28] Md Muksitul Haque, Phylogeny++:A Software System for Phylogenetic Tree Simulation based on Complex and Mixed Models CS 674 Software Architecture Fall 2007

[29] Pressman, Roger S., “Software Engineering, A Practitioner’s Approach”, Mcgraw-Hill, New-York, 1992.

[30] Glade - a User Interface Designer for GTK+ and GNOME http://glade.gnome.org/

[31] Maddison DR, Swofford DL, Maddison WP., NEXUS: an extensible file format for systematic information. http://www.ncbi.nlm.nih.gov/pubmed/11975335

[32] Steward, D.,V., “Software Engineering with Systems Analysis and Design”, Brooks/Cole publishing Company, Monterey, California, 1987.

[33] Shu Li, Development of Parallel Simulation Software for the Study of Thermal Process in Kuparuk River Basin in Alaska, Thesis, UAF, August 1996.

APPNEDIX A: SOFTWARE PROJECT PLAN

I. Introduction

This document is the project plan. A phylogenetic tree simulator PyloSim is a C program to be developed for Linux and MacOSx machines. It simulates the phylogenetic inference scheme using different mathematical models.

II. Project Objectives

1. Objectives:
   A. Simulator with 27 different mathematical models for simulating the phylogeny inference scheme
   B. A simulator that would allow creation of newer models with variable input and custom bases
   C. Easy to use GUI with default template for each models
2. Major functions:
   A. Parser that parses input data to the models requirement
   B. GUI model that helps user by giving easy to use and default template and allows user to view output data
   C. Mathematical Model loader allows existing and new custom and complex model to be created as given input by the user.

III. Main project risks

1. Insufficient main memory space
2. Insufficient disk space

   The above two risk occur because of existing data requirements for larger trees and large number of bases.

IV. Schedule

1. Work Breakdown Structure (Appendix B)
2. Task Network – PERT (Appendix C)
3. Gantt Chart (Appendix D)

V. Project Resources

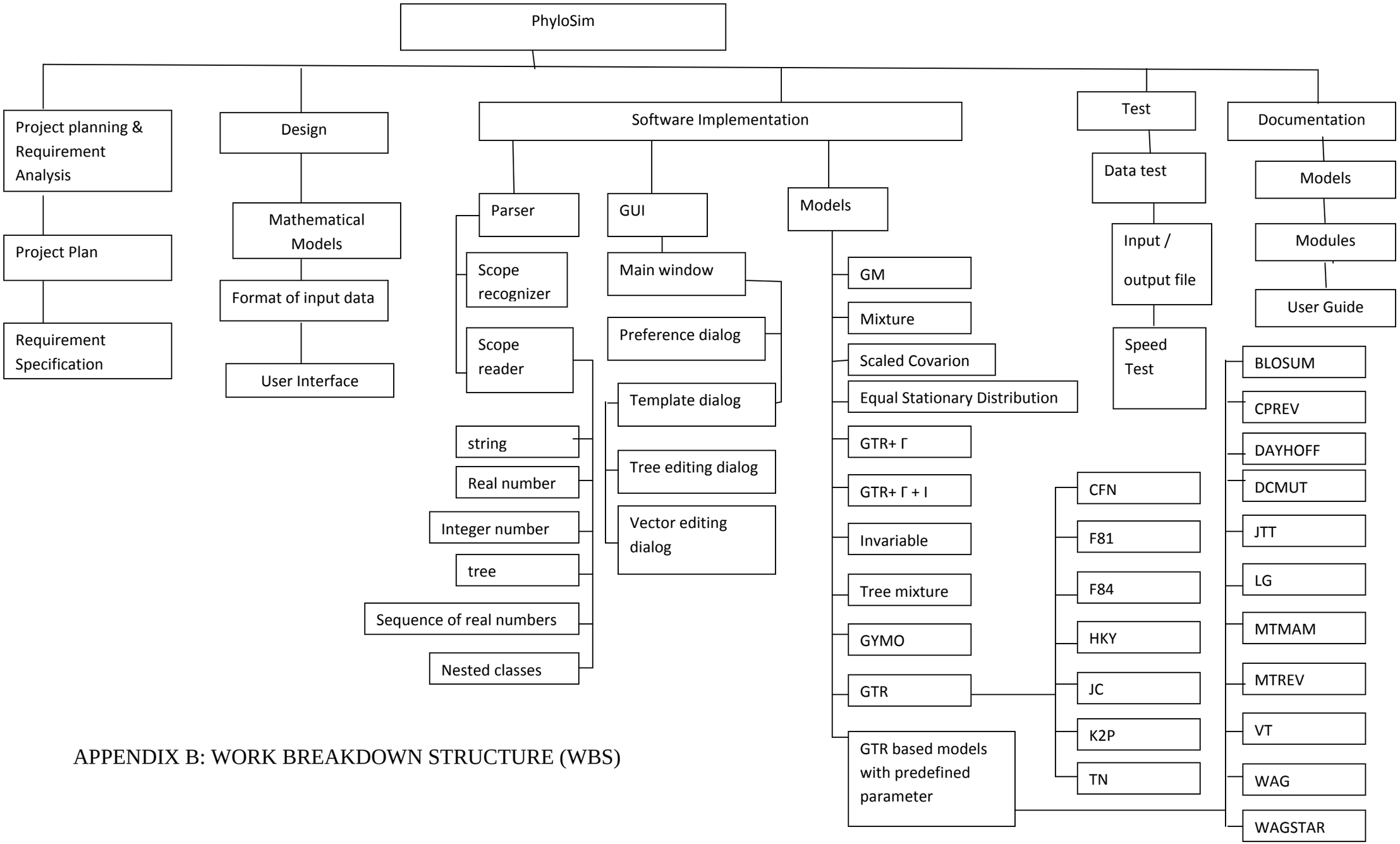
1. Software: gcc, gsl, gtk, glade
2. Hardware: IBM T60

VI. Tracking and Control Mechanisms

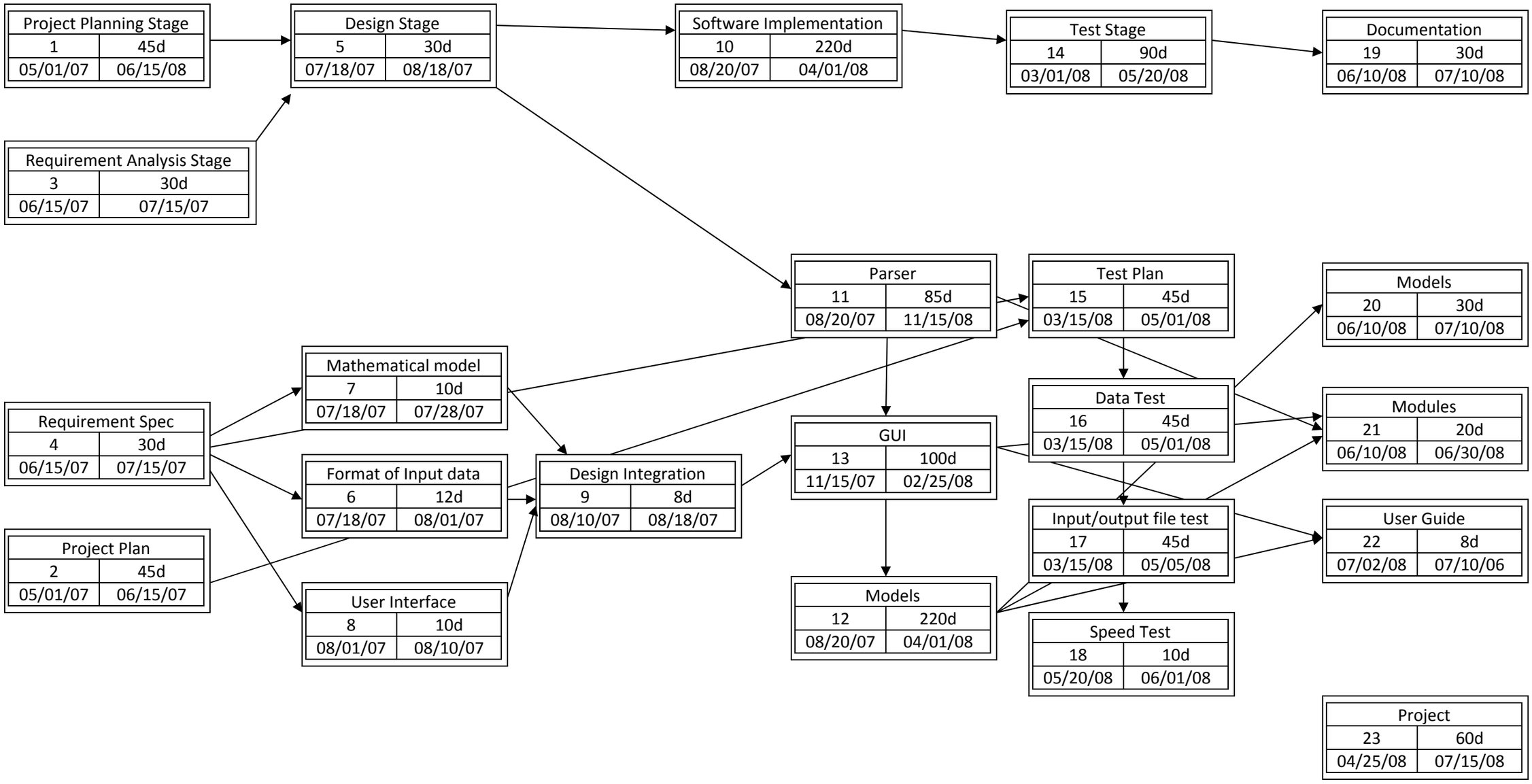
WBS PERT and Gantt chart are the basic tracking and control mechanism.

VII. Appendices

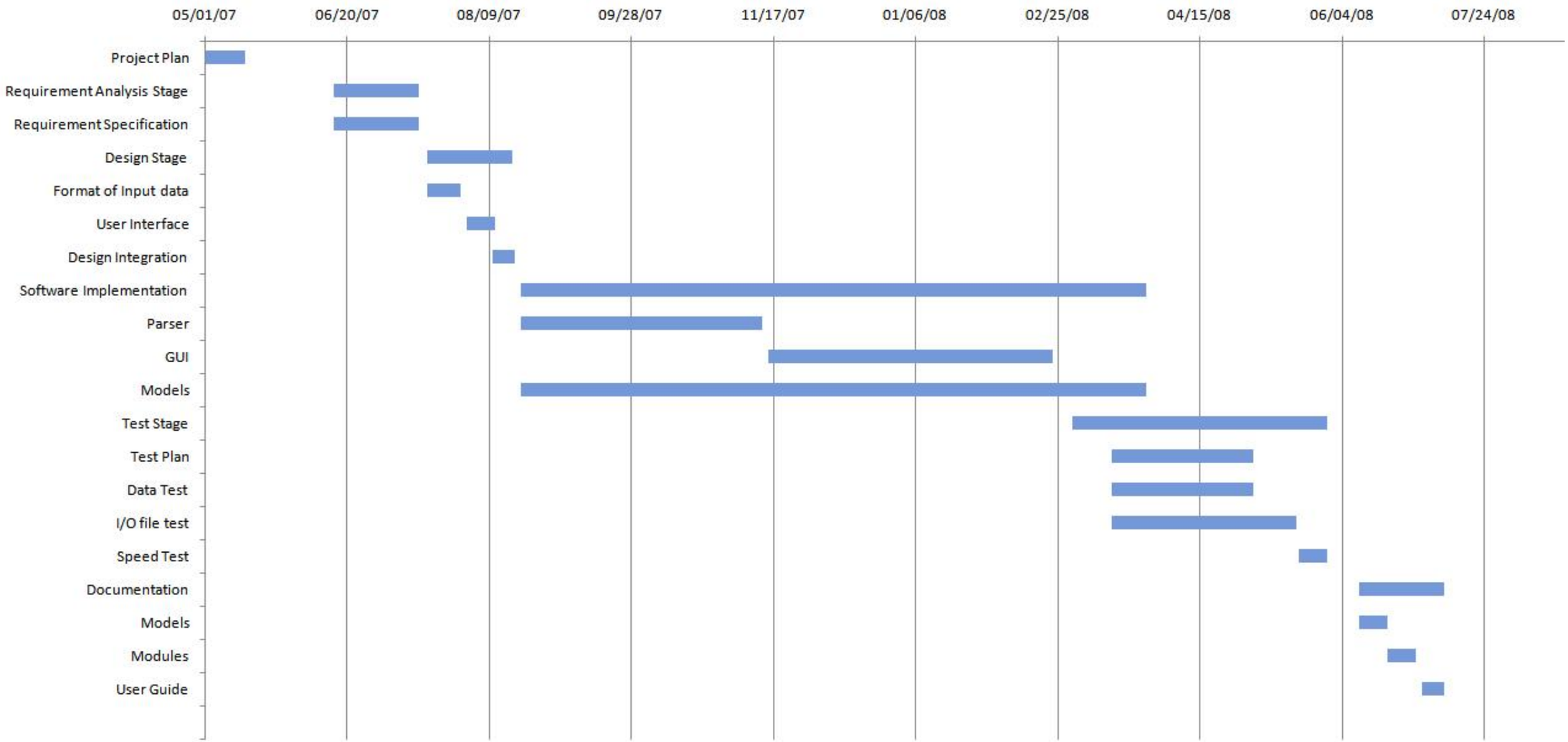
WBS, PERT, Gantt.



APPENDIX B: WORK BREAKDOWN STRUCTURE (WBS)



Appendix C: Task Network –PERT Chart



Appendix D: Gantt Chart

APPENDIX E: TEST DOCUMENTATION

Methods used for testing various models

During the development of the software, each model was tested as it was implemented. Methods of testing varied with the model, depending on the complexity of the model and the availability of other software to make inferences of parameters from the sequences produced by PhyloSim.

For special choices of parameters of the GM model, the form of the sequence output could be easily predicted. For instance, on a tree relating 6 taxa, one edge could be chosen to have some non-identity Markov matrix describing substitutions on it, with all other edges having the Markov matrices set to be the identity. Then all sequences for taxa on one side of the chosen edge would have the same sequences, and similarly for the other side. By comparing a sequence from one side to one from the other, it was possible to recover an approximate value for the Markov matrix, much as was done in the description of the GM model in an earlier chapter of this project report. Using similar ideas, it was possible to test the GM model thoroughly.

For the GTR model and many variants such as JC, Kimura, HKY, GTR+Gamma+I, additional testing was done using PAUP*. Sequences could be simulated for some parameter choice, and then PAUP* could be used to infer the parameters from the sequences. For long sequences, the inferred parameters would closely match the ones used to generate the simulation, as random probabilistic errors would be small. For shorter sequences, the errors would be larger, as predicted by statistical theory. All though this is a probabilistic confirmation that PhyloSim behaves correctly, that is the best that can be done since it implements a probabilistic model.

For covarion models, mixture models, and tree-mixture models, the output of PhyloSim was compared with that of limited implementations of sequence simulators in MATLAB. Again, the output from PhyloSim behaved as it should statistically, offering better and better agreement as the sequence length grew longer.

After the development of the software, more extensive testing for all the models was performed by Dr. Elizabeth Allman. As Dr. Elizabeth had not been involved in any of the initial testing, this provided a strong independent test. While her testing methods were similar, she performed simulations with a broader range of parameter values and trees and used both PAUP* and PhyML for inference since these share no code, and wrote new MATLAB codes to independently confirm many of the calculations done as intermediate steps in PhyloSim.

Finally, it should be noted that the structure of the program PhyloSim is such that testing one model actually tests features of many. For instance, all model implementations require use of most of the GM code, so that confirming the GTR model is implemented correctly provides evidence that GM is as well.

The Simulation results obtained from Dr. Elizabeth Allman is tests of PhyloSim are shown in the next page.

SIMULATION CHECK

Model	Seq-gen	PhyloSim	comments
JC		Okay	
K2P			GTR vs K2P analysis – check titvratio consistent with sg results
F81		Okay	
F84		Okay	should check with PAUP for value of k
HKY		Okay	should check with PAUP for value of titv
TN			look into phyml (and other) parameterizations
CFN			
GTR	okay	Okay	Sg branch lengths are scaled by P_{var} see GTR+I comment
GTR+I	Okay	Okay	
GTR+G4+I	Okay	Okay	
GTR+G4	okay	Okay	
JTT	okay	Okay,phyml on 4 taxarecovers tree	Max(Psg-Pphylo)~0.0015
WAG		phyml on 4-taxa recovers tree	Max(Psg-Pphylo)~0.0025
BLOSUM		phyml on 4-taxon tree is accurate	Max(Psg-Pphylo)~0.0018
mtREV		phyml on 4-taxon tree is accurate	
WAG*	okay	WAG and user input ok	
PAM			
cpREV			
DAYHOFF		phyml on 4-taxa recovers tree	
DCMut		phyml on 4-taxa recovers tree	
mtMAM		phyml on 4-taxa recovers tree	
VT		phyml on 4-taxa recovers tree	
LG		phyml on 4-taxa recovers tree	
codon			

Dr. Allman's Test Results

APPENDIX F: Sample Source Code

Source Code of Tree GUI

```
static void dialog_treeedit_setup (GtkWidget * win, void * user_data) {
	GtkWidget * mt = intf_get_widget (win, "layout_main");
	GtkWidget * elab = intf_get_widget (win, "entry_label");
	GtkWidget * eval = intf_get_widget (win, "entry_value");
	draw_data_t * dd = g_new (draw_data_t, 1);
	char * tree = user_data;
	GNode * root;

	if (strlen (tree)) {
		parser_scan_tree (tree, &root);
		dd->tree = g_node_copy_deep (root, make_draw_node, dd);
		parser_free_tree (root);
	} else {
		dd->tree = g_node_new (make_draw_node (NULL, NULL));
	}
	dd->border = 10;
	dd->cell_width = 30;
	dd->cell_height = 30;
	dd->dialog = win;
	dd->layout = GTK_LAYOUT (mt);
	dd->max_width = 0;
	setup_draw_data (dd);
	g_object_set_data (G_OBJECT (win), "draw_data", dd);
	move_widgets (GTK_LAYOUT (mt), dd->tree, dd);
	if (dd->cell_width < dd->max_width) {
		dd->cell_width = dd->max_width;
		dd->width = (dd->max_x - dd->min_x + 1) * dd->cell_width + 2 * dd->border;
		dd->height = (dd->max_y + 1) * dd->cell_height + 2 * dd->border;
		gtk_layout_set_size (dd->layout, dd->width, dd->height);
		move_widgets (GTK_LAYOUT (mt), dd->tree, dd);
		gtk_widget_queue_draw_area (GTK_WIDGET (dd->layout), 0, 0, dd->width, dd->height);
	}
	gtk_widget_show_all (mt);
	g_node_traverse (dd->tree, G_IN_ORDER, G_TRAVERSE_ALL, -1, find_select_node, ((draw_node_data_t *) (dd-
>tree->data))->wid);
	g_signal_connect (G_OBJECT (mt), "expose_event",  G_CALLBACK (te_lo_expose_cb), dd);
	g_signal_connect (G_OBJECT (elab), "changed",  G_CALLBACK (te_current_label_changed), dd);
	g_signal_connect (G_OBJECT (eval), "changed",  G_CALLBACK (te_current_value_changed), NULL);
	g_signal_connect (G_OBJECT (win), "destroy",  G_CALLBACK (dialog_treeedit_destroy_cb), dd);
}

static void tree_add_node_cb (GtkButton * button, gpointer user_data) {
	GtkWidget * dialog = intf_get_widget (GTK_WIDGET (button), "dialog_treeedit");
	draw_data_t * dd = g_object_get_data (G_OBJECT (dialog), "draw_data");
	GNode * node = g_object_get_data (G_OBJECT (dialog), "cur_node");
	draw_node_data_t * nd = g_new (draw_node_data_t, 1);
	GNode * ch;

	nd->wid = NULL;
	nd->label = g_strdup ("<new>");
	nd->value = g_strdup ("<new>");
	ch = g_node_new (nd);
	g_node_append (node, ch);
	nd = g_new (draw_node_data_t, 1);
	nd->wid = NULL;
```

```
	nd->label = g_strdup ("<new>");
	nd->value = g_strdup ("<new>");
	ch = g_node_new (nd);
	g_node_append (node, ch);

	setup_draw_data (dd);
	dd->max_width = 0;
	move_widgets (dd->layout, dd->tree, dd);
	if (dd->cell_width < dd->max_width) {
		dd->cell_width = dd->max_width;
		dd->width = (dd->max_x - dd->min_x + 1) * dd->cell_width + 2 * dd->border;
		dd->height = (dd->max_y + 1) * dd->cell_height + 2 * dd->border;
		gtk_layout_set_size (dd->layout, dd->width, dd->height);
		move_widgets (dd->layout, dd->tree, dd);
		gtk_widget_queue_draw_area (GTK_WIDGET (dd->layout), 0, 0, dd->width, dd->height);
	}
	g_node_traverse (dd->tree, G_IN_ORDER, G_TRAVERSE_ALL, -1, find_select_node, ((draw_node_data_t *) (node-
>data))->wid);
	gtk_widget_queue_draw_area (GTK_WIDGET (dd->layout), 0, 0, dd->width, dd->height);
}

static void tree_remove_node_cb (GtkButton * button, gpointer user_data) {
	GtkWidget * dialog = intf_get_widget (GTK_WIDGET (button), "dialog_treeedit");
	draw_data_t * dd = g_object_get_data (G_OBJECT (dialog), "draw_data");
	GNode * node = g_object_get_data (G_OBJECT (dialog), "cur_node");
	int w = dd->width;
	int h = dd->height;

	g_node_children_foreach (node, G_TRAVERSE_ALL, free_draw_node_data, NULL);
	g_node_children_foreach (node, G_TRAVERSE_ALL, (GNodeForeachFunc) g_node_destroy, NULL);
	setup_draw_data (dd);
	move_widgets (dd->layout, dd->tree, dd);
	g_node_traverse (dd->tree, G_IN_ORDER, G_TRAVERSE_ALL, -1, find_select_node, ((draw_node_data_t *) (node-
>data))->wid);
	gtk_widget_queue_draw_area (GTK_WIDGET (dd->layout), 0, 0, w, h);
	gtk_widget_queue_draw_area (gtk_widget_get_parent (GTK_WIDGET (dd->layout)), 0, 0, w, h);
}

static void button_fit_clicked_cb (GtkButton * button, gpointer user_data) {
	GtkWidget * dialog = intf_get_widget (GTK_WIDGET (button), "dialog_treeedit");
	draw_data_t * dd = g_object_get_data (G_OBJECT (dialog), "draw_data");

	dd->max_width = 0;
	move_widgets (dd->layout, dd->tree, dd);
	dd->cell_width = dd->max_width + 10;
	dd->width = (dd->max_x - dd->min_x + 1) * dd->cell_width + 2 * dd->border;
	dd->height = (dd->max_y + 1) * dd->cell_height + 2 * dd->border;
	gtk_layout_set_size (dd->layout, dd->width, dd->height);
	move_widgets (dd->layout, dd->tree, dd);
	gtk_widget_queue_draw_area (GTK_WIDGET (dd->layout), 0, 0, dd->width, dd->height);

}

static void te_current_label_changed (GtkEntry * entry, gpointer user_data) {
	GtkWidget * dialog = intf_get_widget (GTK_WIDGET (entry), "dialog_treeedit");
	GNode * node = g_object_get_data (G_OBJECT (dialog), "cur_node");
	draw_node_data_t * nd = node->data;
	GtkLabel * lab = GTK_LABEL (gtk_bin_get_child (GTK_BIN (gtk_bin_get_child (GTK_BIN (nd->wid)))));
	GtkRequisition req;
	draw_data_t * dd = user_data;
	const char * text = gtk_entry_get_text (entry);
```

```
	gtk_label_set_text (lab, text);
	gtk_widget_size_request (nd->wid, &req);
	if (req.width > dd->max_width) {
		dd->cell_width = 30 + req.width;
		dd->max_width = 30 + req.width;
		dd->width = (dd->max_x - dd->min_x + 1) * dd->cell_width + 2 * dd->border;
		dd->height = (dd->max_y + 1) * dd->cell_height + 2 * dd->border;
		gtk_layout_set_size (dd->layout, dd->width, dd->height);
		move_widgets (dd->layout, dd->tree, dd);
		gtk_widget_queue_draw_area (GTK_WIDGET (dd->layout), 0, 0, dd->width, dd->height);
	}
	g_free (nd->label);
	nd->label = g_strdup (text);
}

static void draw_tree (GtkLayout * lo, GNode * n, draw_data_t * dd) {
	GNode * nc = NULL;
	draw_node_data_t * nd, * ndn;
	int xc, yc, xcn, ycn;

	if (!n) n = dd->tree;
	nd = n->data;
	xc = dd->border + (nd->x - dd->min_x) * dd->cell_width;
	yc = dd->border + dd->cell_height * nd->y;

	yc += nd->wid->allocation.height / 2;
	xc += nd->wid->allocation.width / 2;
	if ((nc = g_node_nth_child (n, 0))) {
		ndn = nc->data;
		xcn = dd->border + (ndn->x - dd->min_x) * dd->cell_width;
		ycn = dd->border + dd->cell_height * ndn->y;

		ycn += ndn->wid->allocation.height / 2;
		xcn += ndn->wid->allocation.width / 2;
		gdk_draw_line (lo->bin_window, GTK_WIDGET (lo)->style->fg_gc[GTK_STATE_ACTIVE], xc, yc, xcn,
ycn);
		draw_tree (lo, nc, dd);
	}
	if ((nc = g_node_nth_child (n, 1))) {
		ndn = nc->data;
		xcn = dd->border + (ndn->x - dd->min_x) * dd->cell_width;
		ycn = dd->border + dd->cell_height * ndn->y;

		ycn += ndn->wid->allocation.height / 2;
		xcn += ndn->wid->allocation.width / 2;
		gdk_draw_line (lo->bin_window, GTK_WIDGET (lo)->style->fg_gc[GTK_STATE_ACTIVE], xc, yc, xcn,
ycn);
		draw_tree (lo, nc, dd);
	}
}

static gboolean te_lo_expose_cb (GtkWidget * widget, GdkEventExpose * event, gpointer data) {
	draw_tree (GTK_LAYOUT (widget), NULL, data);
	return FALSE;
}

static gboolean free_draw_node (GNode *node, gpointer data) {
	draw_node_data_t * nd = node->data;
	g_free (nd->value);
	g_free (nd->label);
	g_free (nd);
	return FALSE;
```

```
}

static void select_node (GNode *node, draw_data_t * dd) {
	draw_node_data_t * nd = node->data;
	GtkWidget * add = intf_get_widget (dd->dialog, "button_add");
	GtkWidget * rem = intf_get_widget (dd->dialog, "button_remove");
	GtkWidget * eval = intf_get_widget (dd->dialog, "entry_value");
	GtkWidget * elab = intf_get_widget (dd->dialog, "entry_label");
	GtkWidget * bval = intf_get_widget (dd->dialog, "vbox_value");
	GtkWidget * blab = intf_get_widget (dd->dialog, "vbox_label");
	int isl = G_NODE_IS_LEAF (node);

	g_signal_handlers_block_matched (elab, G_SIGNAL_MATCH_FUNC, -1, 0, NULL, te_current_label_changed,
NULL);
	g_signal_handlers_block_matched (eval, G_SIGNAL_MATCH_FUNC, -1, 0, NULL, te_current_value_changed,
NULL);
	if (node->parent) {
		gtk_widget_set_sensitive (bval, isl);
		gtk_widget_set_sensitive (blab, TRUE);
		gtk_entry_set_text (GTK_ENTRY (eval), nd->value ? nd->value : "");
		gtk_entry_set_text (GTK_ENTRY (elab), nd->label);
	} else {
		/* root node */
		gtk_widget_set_sensitive (bval, FALSE);
		gtk_widget_set_sensitive (blab, FALSE);
		gtk_entry_set_text (GTK_ENTRY (eval), "");
		gtk_entry_set_text (GTK_ENTRY (elab), "");
	}
	if (isl) {
		gtk_widget_set_sensitive (add, TRUE);
		gtk_widget_set_sensitive (rem, FALSE);
	} else {
		gtk_widget_set_sensitive (add, FALSE);
		gtk_widget_set_sensitive (rem, TRUE);
	}
	g_object_set_data (G_OBJECT (dd->dialog), "cur_node", node);
	g_signal_handlers_unblock_matched (elab, G_SIGNAL_MATCH_FUNC, -1, 0, NULL, te_current_label_changed,
NULL);
	g_signal_handlers_unblock_matched (eval, G_SIGNAL_MATCH_FUNC, -1, 0, NULL, te_current_value_changed,
NULL);
}

static void dialog_treeedit_destroy_cb (GtkObject *object, gpointer user_data) {
	draw_data_t * dd = user_data;

	g_node_traverse (dd->tree, G_IN_ORDER, G_TRAVERSE_ALL, -1, free_draw_node, NULL);
	g_node_destroy (dd->tree);
	g_free (dd);
}

static gpointer make_draw_node (gconstpointer src, gpointer data) {
	draw_node_data_t * nd = NULL;
	const node_data_t * node = src;

	nd = g_new (draw_node_data_t, 1);
	nd->wid = NULL;
	if (node) {
		nd->label = g_strdup (node->label);
		nd->value = node->value ? g_strdup (node->value) : NULL;
	} else {
		nd->label = NULL;
		nd->value = NULL;
```

```
        }
        nd->x = -1;
        nd->y = -1;
        return nd;
}
```

Appendix G: Program Snapshots

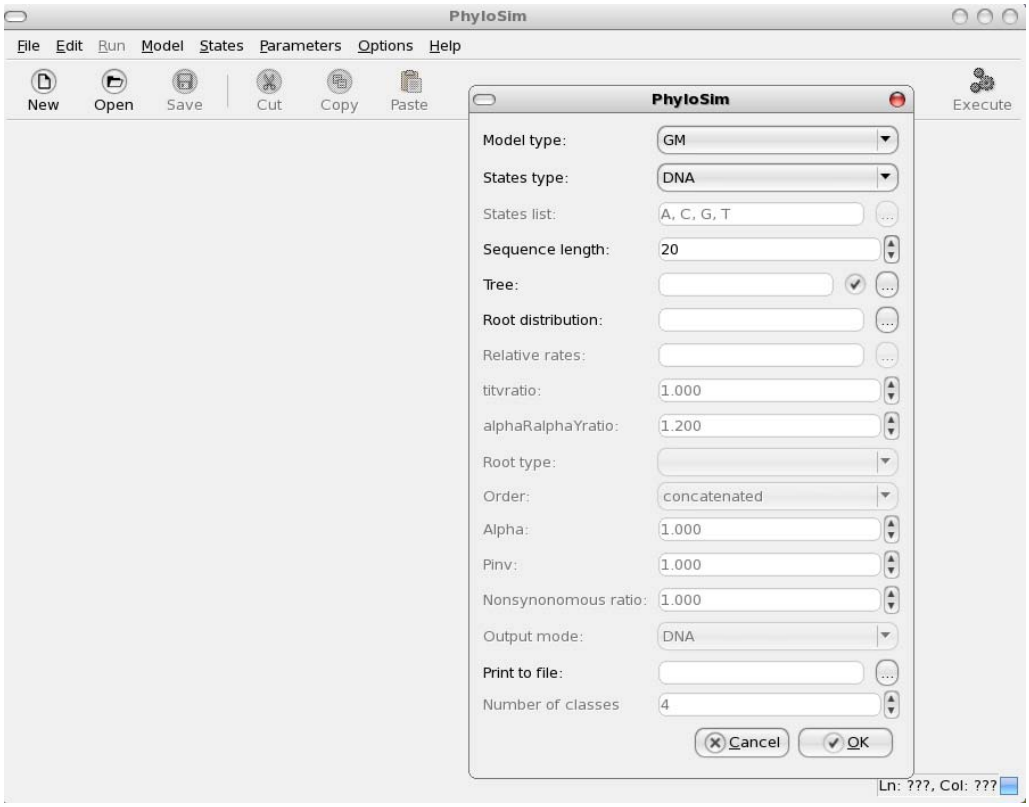


Fig 1: Shows the basic template where model information is to be given

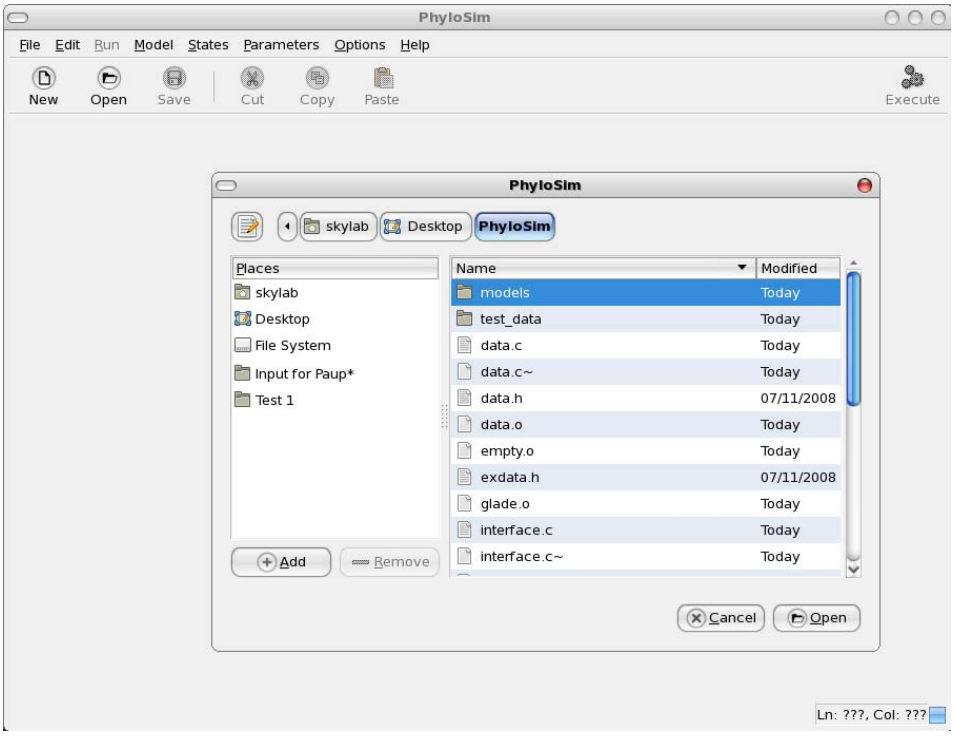


Fig 2: User finds the input file from any location in the computer

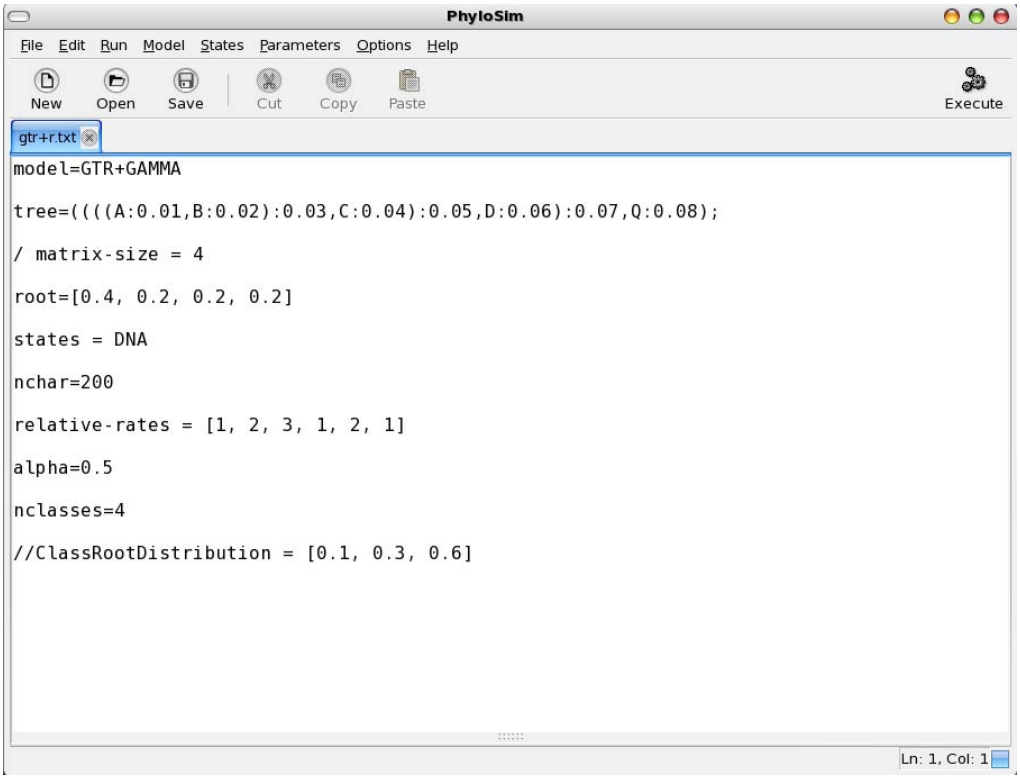


Fig 3: User loads the input file for processing

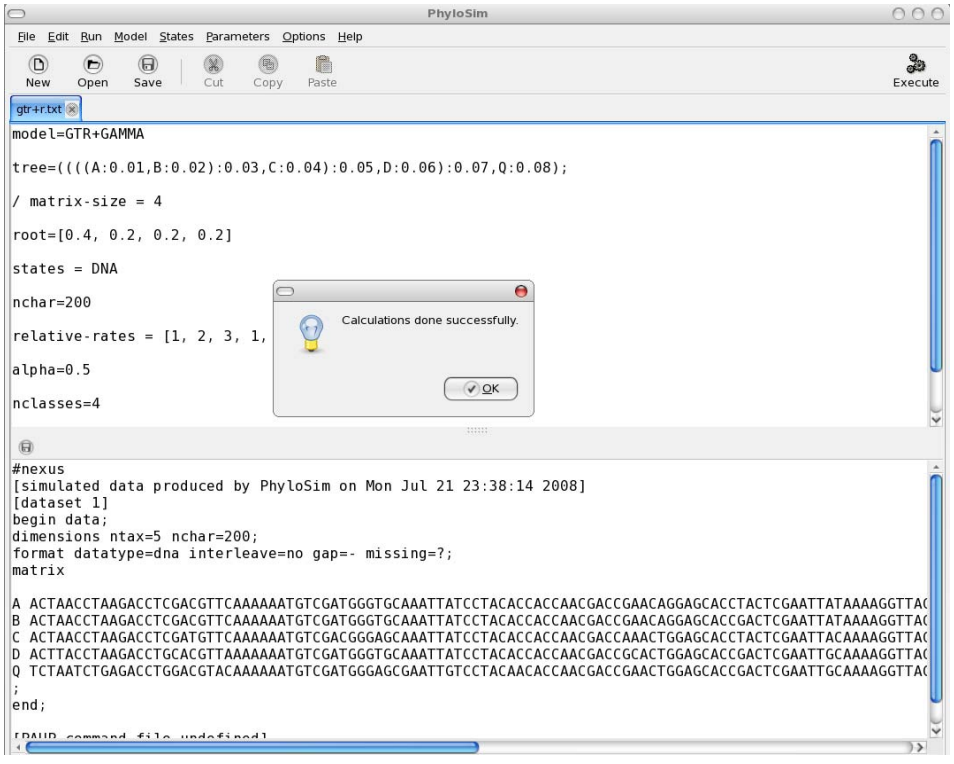


Fig 4: Shows the program has executed successfully

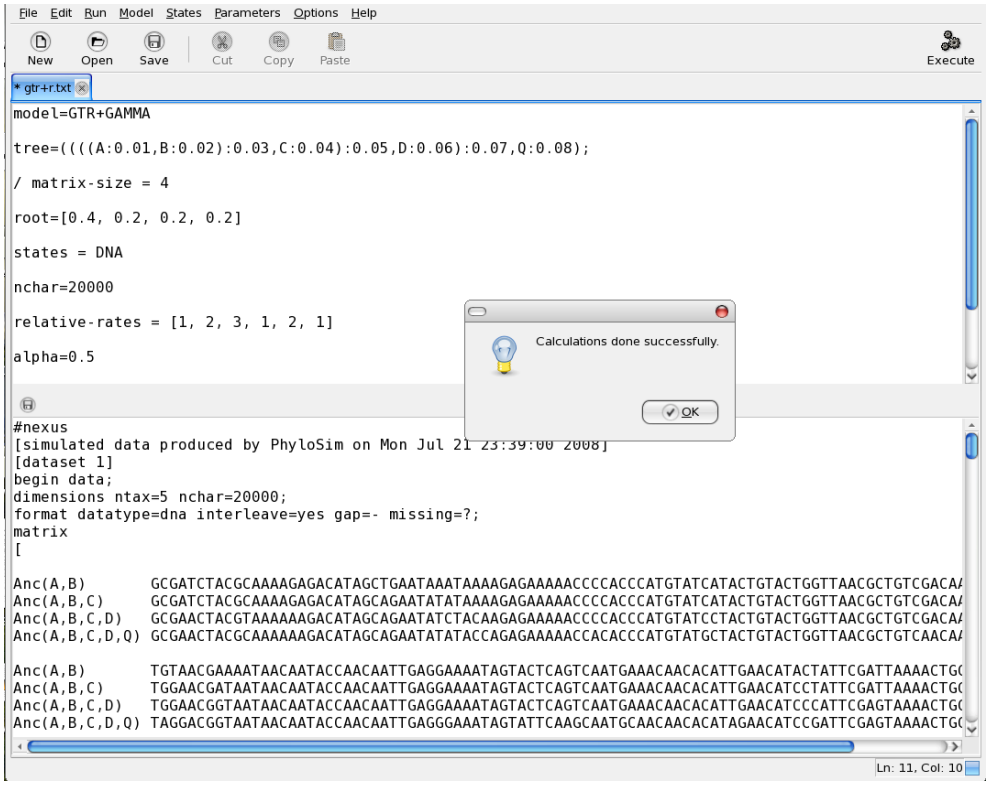


Fig 5: Shows the output below the input with Ancestral sequence

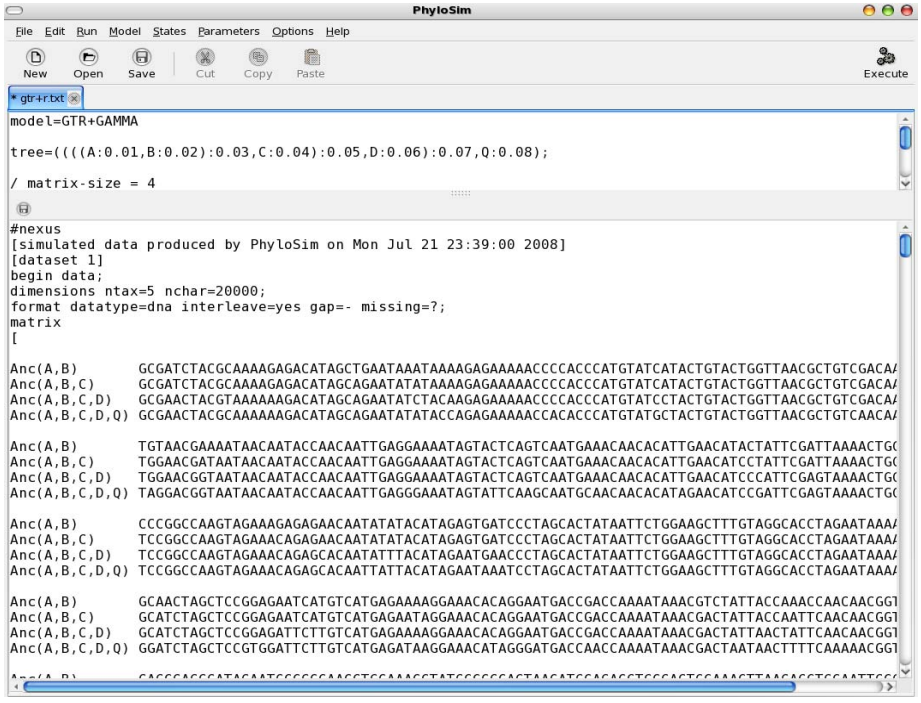


Fig 6: PhyloSim output shown with Ancestral sequence in interleaved form

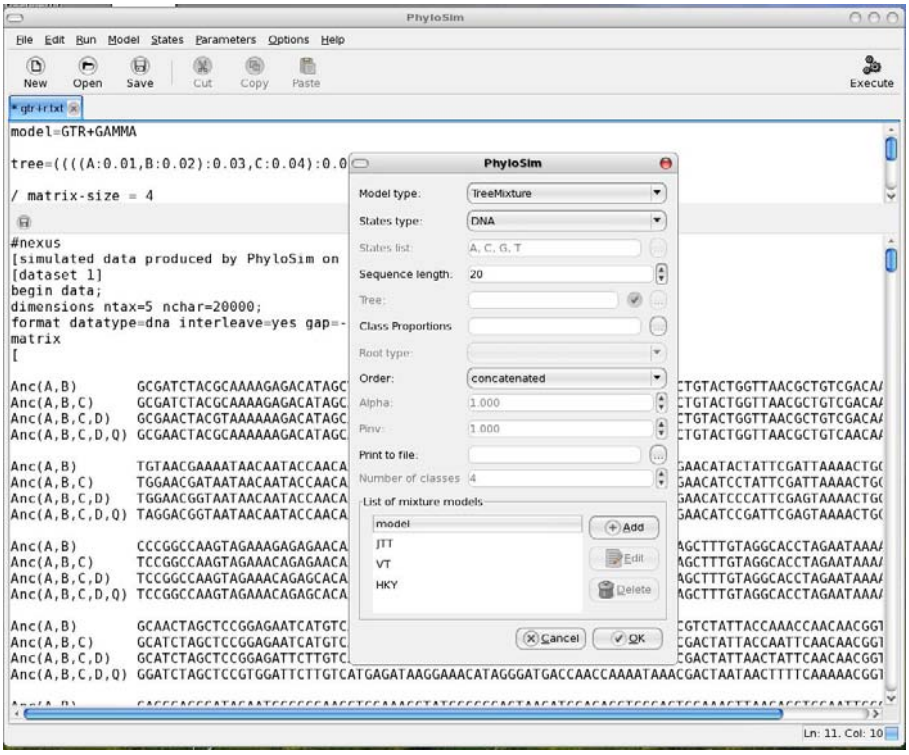


Fig 7: Shows the basic template of TreeMixture model which is a mixture of 3 other models here

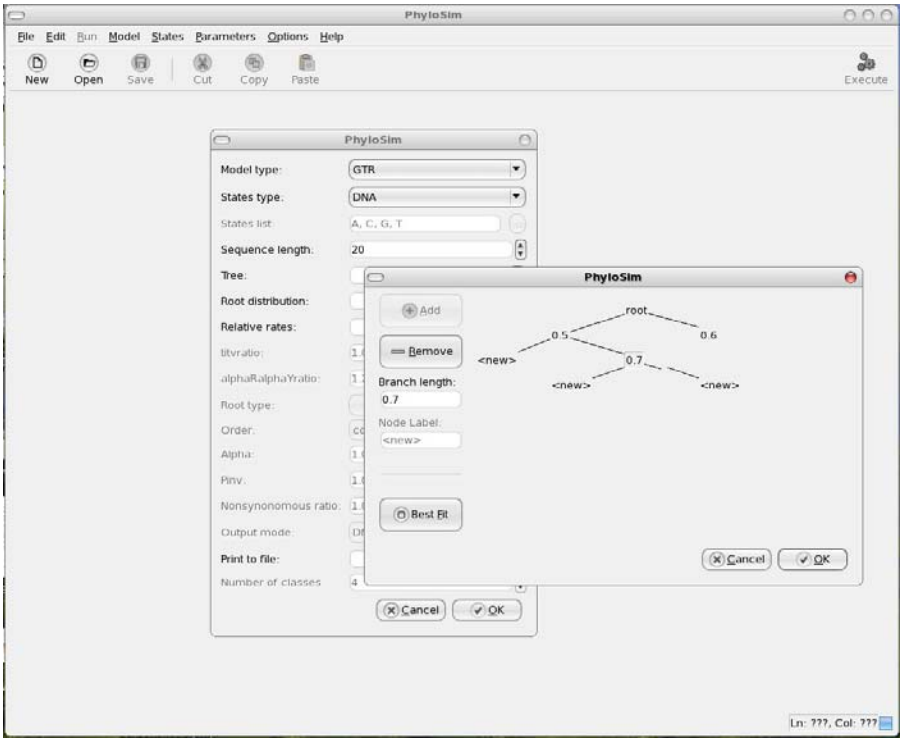


Fig 8: Shows the easy to use tree editor, entering branch length and Node label

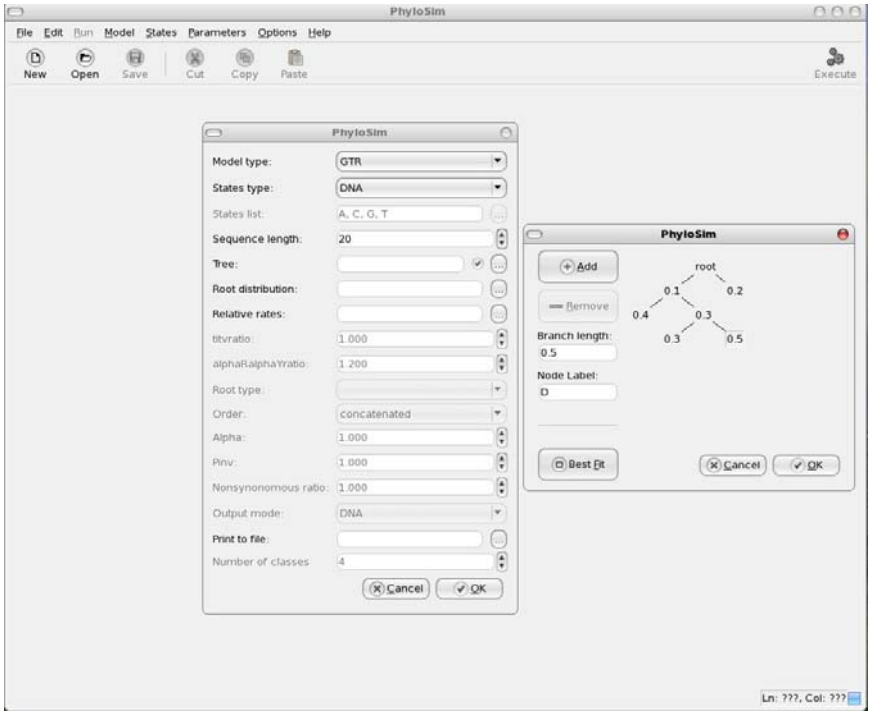


Fig 9: Shows the GUI tree editor for a different model

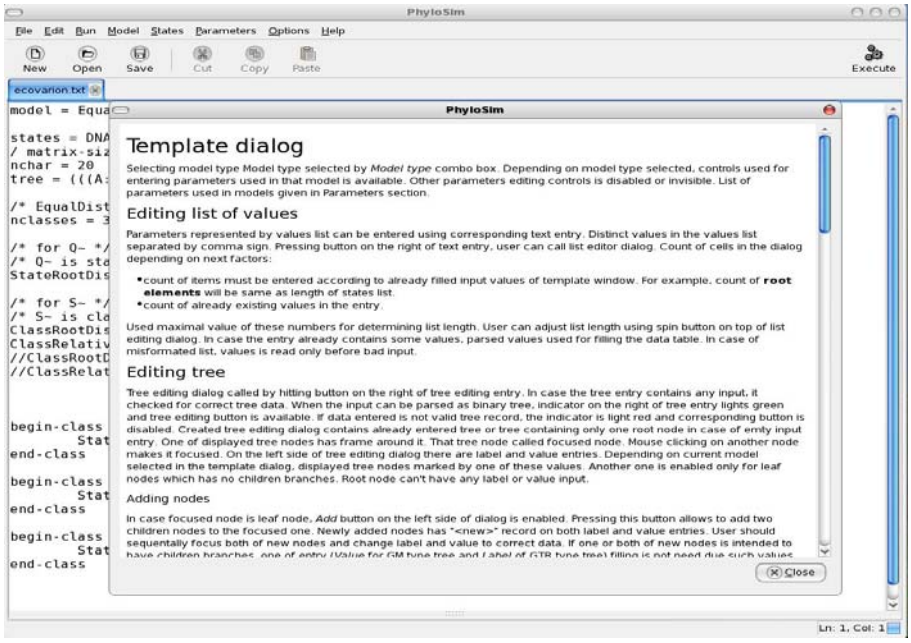


Fig 10: Shows the Help Menu with Template dialog how to

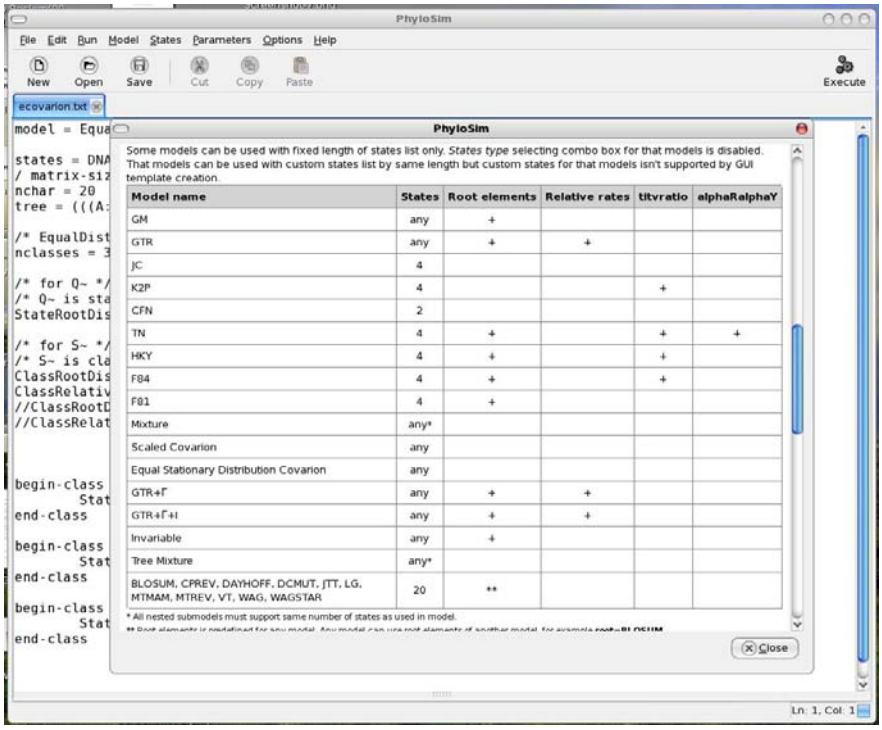


Fig 11: Shows the help menu with which model needs which parameters