THE STATE COMMITTEE FOR COMMUNICATION, INFORMATIZATION AND TELECOMMUNICATION TECHNOLOGIES OF THE REPUBLIC OF UZBEKISTAN

TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES

Manuscript rights

UDC 004.4'2:61

ERALIYEV JAVLON RUSTAMBEKOVICH

Development of algorithmic procedures for the analysis of phylogenetic data

5A330201 - Computer systems and software

In partial fulfillment of

the requirements for the degree of

Masters of Sciences

Research supervisor:

Hudayberdiyev M.X.

Tashkent - 2014

CONTENTS

Introduction

I want to start this thesis with the words of I.A. Karimov and thank him for all what we have. In one of his works "Uzbekistan: its way of renewal and progress," he gives the following definition: "Uzbekistan - the state of a great future. It is a sovereign, democratic state. This is a state based on the principles of humanism, ensuring the rights and freedoms of citizens, regardless of nationality, religion, social status or political convictions." [1] In the same paper, he emphasizes that the source of state power is the people. It will determine the policy of the state, which should be aimed at the welfare of the individual and society, decent life for all citizens of Uzbekistan [2].

Phylogenetic is the study of the evolutionary histories of living organisms, and represent the evolutionary divergences by finite directed (weighted) graphs, or directed (weighted) trees, known as *phylogeny*. Based on molecular sequences, phylogenetic trees can be built to reconstruct the evolutionary tree of species involved. In particular, the representation derived from genes or protein sequences is known as *gene phylogeny*, while the representation of the evolutionary path of the species are often referred as species *phylogeny*. It only describes the evolution of a particular gene or encoded protein, and this sequence could evolve much more or less differently than other genes in the genome, or it may have completely different evolutionary history from the rest of the genome due to horizontal gene transfers. Therefore, the evolution of a particular gene only provides a local picture, not necessarily reflect the *global* evolutionary picture of the species. We can only hope that we could assemble the jigsaw puzzle pieces with a wide selection of gene family to give an overall assessment of the species evolution.

While in general the topology in phylogenetic trees represents the relationships between the taxa, assigning scales to edges in the trees could provide extra information on the amount of evolution divergence as well as the time of

the divergence. The phylogenetic trees with the scaled edges are called *phylograms*, while the non-scaled phylogenetic trees are called *cladograms*. Purely for the sake of computers data processing, some special formats were artificially created, such as Newick format. From biological point of view, the building of phylogenetic trees can be roughly divided into the following steps.

Motivation. The root of the problem lies in the lack of a certain mechanism for analyzing phylogenetic sequences. This makes it possible to analyze and build phylogenetic trees from given data. The problem is to optimize constructing large trees using algorithms of estimate calculations.

The aim is to find an optimal solution in a set of candidate solutions. There are other types of problems besides optimisation and decision problems. This dissertation will deal with optimisation problems.

Problems are not only categorised on the basis of the types of answer they seek, but also on the basis of how complex it is to compute the answers. For some problems, there is no algorithm giving always the correct answer. For most of the problems encountered in practice, however, the complexity of their solution is measured in terms of the amount of computational resources (typically time and memory space) that algorithms need in order to compute the correct output.

Research object. Phylogeny, phylogenetic tree and phylogenetic sequences.

Research subject. Methods of phylogenetic analysis, phylogenetic analysis algorithms, phylogenetic analysis software are the object of research.

Research question. The following research questions were defined from given below background of the problem:

How can be used algorithms of estimate calculations in phylogeny analyzing systems?;

Can algorithms of estimate calculations improve efficiency of phylogeny analizing systems?;

Research goals. Learn and analyze phylogenetic analysis systems and softwares;

Find main characteristics and ways for improving phylogenetic analysis;

Define methods and concepts for phylogenetic data analysis;

Develop process for phylogenetic data analysis;

Integrate algorithms of estimate calculations into phylogenetics;

Develop phylogenetic data analysis software;

Research hypothesis. Algorithms of estimate calculation are more efficient to reconstruct phylogenetic tree.

Existing researches. There are many foregn researches and articles devoted to the phylogenetics. Scientists use results of the researches in real world projects and analysis . For example, in biology, in medicine and etc.

Developed many programms such as PHYLIP, PAUP, MacClade, TREECON, Spectrum, TREEMAP, DNA Stacks, GeneDoc, Seq-Gen, Phylodendron, GeneTree, PASSML, BioEdit, MEGA, PAST, ClustalW and etc.

Found that developed products have many advantages and some disadvantages. Main disadvantages are related with efficiency. No product which uses algorithms of estimate calculations.

Researches methods. Systematic analysis, cluster analysis, programming techniques and methods, modelling, benchmarking and comparative methods are used.

Theoretical and practical importance. Developed software can be used by scientists in biology. Implemented methods are more efficient for reconstructing phylogenetic trees.

Benefit for the Science and contributions. Analized pro and cons of available phylogenetic tree building systems;

Improved a process and methods of phylogenetic tree reconstructing;

Developed software using the process and the methods;

Content. The Thesis includes introduction, 3 chapters, each chapter has a introduction and summary, conclusion, references and applications. Main part of the thesis wrote in 48 pages.

Chapter 1. General concept and analysis of existing methods for the analysis of phylogenetic data

1. General concepts of phylogeny

The branching pattern of ancestor/descendant relationships among species or their parts (e.g., genes) is a phylogeny. Researchers attempt to estimate these historical relationships by examining character evolution using a tree - a mathematical structure used to model the actual evolutionary history of species or their parts. These inferred trees (historical branching relationships) can be represented as cladograms, where branch lengths are arbitrary and only the branching order is significant, or as phylograms, where the branch lengths are proportional to the amount of evolutionary change along the branch.

Phylogenies were historically used to classify organisms into natural evolutionary groups based on these ancestor/descendant relationships. Indeed, great effort is currently being spent on estimating the Tree of Life to quantify the biodiversity of our planet. However, phylogenies have also spread in use as the utility of the evolutionary framework for numerous other disciplines becomes increasingly obvious. For example, phylogenies are now being extensively used in the biomedical sciences including developmental biology, genomic biology, infectious disease, virology, and human genetics [3].

Phylogenetic Tree. Since this thesis is about phylogenetic trees, it is therefore appropriate to start by defining a tree [4]. Trees can be classified as unrooted or rooted phylogenetic trees. An unrooted phylogenetic tree or just unrooted tree is an acyclic connected graph having no internal vertices of degree two and every leaf having different label. The leaves are vertices of degree one (Figure 1.1).

A rooted tree on tree, on the other hand, is similar to an unrooted tree, except it has one internal vertex of degree two, which is called the root. The

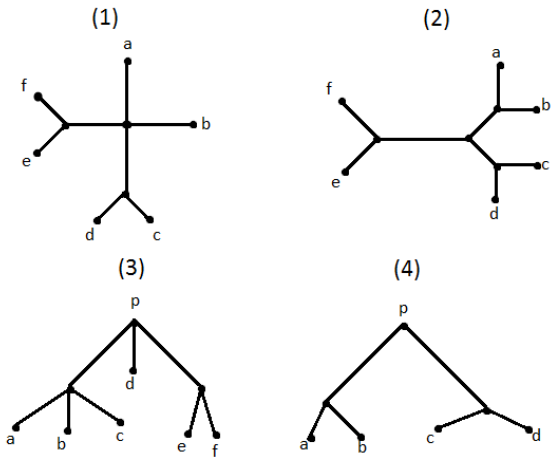

Figure 1.1. Four examples of phylogenetic trees. (1) and (2) are unrooted. (3) and (4) are rooted. (2) and (4) are binary.

internal vertices of unrooted/rooted (except the root) trees can have degree three or greater. For example a binary phylogenetic tree, is a tree having all internal vertices of degree three. Again the only exception is the root, which has degree two (Figure 1.2). In a fully resolved binary phylogenetic tree with n leaf nodes there are n-1 internal nodes.

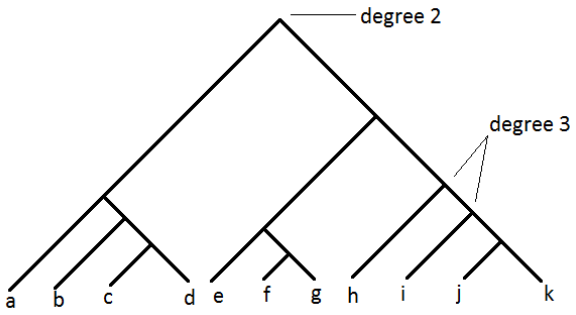

Figure 1.2. In a binary tree each internal node has degree three with the exception of the root which has degree two.

The leaves of the tree represent species. For example let L(T) be the set of leaves for tree T. If T the set of trees, then we can say that L(T) is the union of the leaf sets of the trees in T.

The vertices adjacent to a vertex that are descendants of the vertex are called the children of the vertex, and the adjacent vertex that is an ancestor is called the parent of that vertex. Sometimes the internal vertices of a phylogenetic tree are labelled (section Nested Taxa).

Rooted phylogenetic trees can be displayed with a vertical axis representing the time each branching point occurred. These diagrams are called dendograms (Figure 1.3).

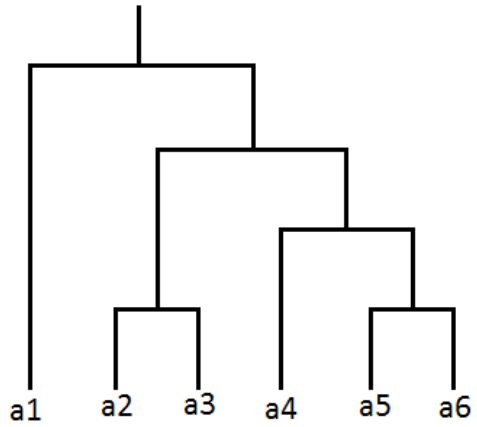

Figure 1.3. An example of dendogram.

Let T be a rooted tree and choose a vertex v in T. If we remove the edge between v and the parent of v, say p, we get two connected subgraphs. Then let v be the root of the subgraphs containing v, then this is called the subtree of T rooted at v. Briefly a subtree T' is a tree whose vertices and edges form the subsets of the vertices and edges of a given tree T. An example of a subtree is shown in Figure 1.4.

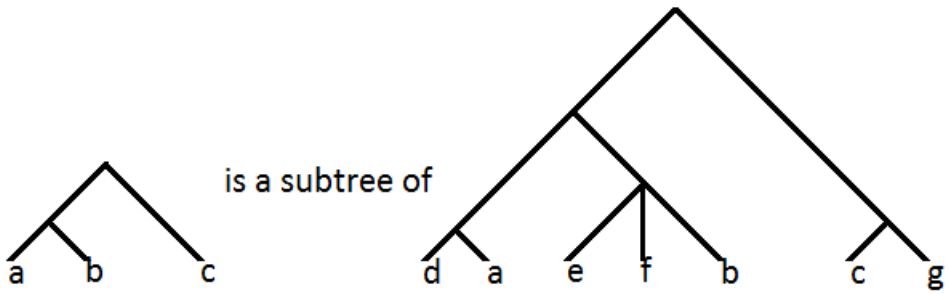

Figure 1.4. Example of a subtree.

For every three leaves a, b, c there are four possible rooted trees with leaf set a, b, c.

The binary rooted trees on three leaves are called rooted triples and ((ab)c) (or ab—c) denotes the rooted triple with a pair of leaves a, b connected to a third

leaf c via the root (Figure 1.5). For a rooted triple ((ab)c) to fit a rooted tree T, the path from a to b does not share any vertices with the path from c to the root. Briefly a rooted triple is a tree with three leaves and two internal vertices.

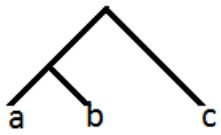

Figure 1.5. A binary rooted triple ((ab)c).

Non-binary rooted trees with three leaves are called fan triples. We call a fan with k leaves a k-fan.

2. Using Algorithms of Estimate Calculations for building distance matrix

Most of the advanced data processing and analysis technologies designed for solving domain-specific problems employ the automation and optimization techniques of decision-making based on 'real' information. The methods of mathematical theory of pattern recognition play an important role here. To carry out image recognition, we need an image representation that corresponds to the requirements of the efficient recognition algorithm chosen for the task [5]. A vast majority of the efficient image recognition algorithms only work with image descriptions or models. To completely use the information contained in images, it is necessary to overcome the principal discrepancy between the nature of images and the data-extraction techniques based on symbol models of images. Thus, there is a practical need for an efficient recognition algorithm that directly deals with images and their fragments. More-over, the algorithm should provide the possibility of posing and solving the problem of choosing the best recognition algorithm. This class of algorithms - algorithms of estimate calculations based on 2D information - was defined by I. Gurevich [6] as a special type of the classical model of the recognition algorithms based on estimate calculations (AEC) introduced by Yu. Zhuravlev [7]. Generally, the

AEC model can cope with the spatial (2D) image structure. In this work, we find these formulas more efficient to reconstruct phylogenetic tree.

The procedures of transforming the initial data in the form convenient for recognition should yield a mathematical model of an image. This model should reflect the inner structure and content of the image as an outcome of operations that construct the image from the subimages and other objects of simpler nature, i.e., from the primitives and objects extracted in the image at different stages of processing. During image recognition, one should use information reflecting the way of pattern formation, i.e., of the image as a whole, and of the objects presented in the image.

Three types of information characterize an image: (i) identifiable objects with a well-defined structure; (ii) identifiable objects with an ill-defined structure; and (iii) nonidentifiable objects. To allow for an image structure means to extract subimages (objects) in an image, to define the possible elementary level for them, and to define the relations between these objects and elements. As a result, the hierarchical structural information of an image may be explicitly presented and utilized. An image is described by a system of objects, each object is described by simpler objects, etc. The structural information can be introduced into recognition process in two ways.

First, according to classical pattern recognition theory, we can use the list of features as a main formalization principle and

(a) two types of features are introduced in the description; they reflect a two-dimensional character of the object to be recognized:

-characteristics which reflect the properties of some local image fragment (the distribution of pixel values in this fragment, the presence or absence of a certain geometrical object in this fragment, the type of the object's shape, etc.);

-characteristics of relations of separate objects and features;

(b) the weights are assigned to the features, which indicate the degree of their importance for image description;

(c) separate features are combined into a system of features and treated as a single feature.

General characterustics of the AEC class. The model of algorithms based on calculation of the estimates (AEC) was successfully used for solving many problems of pattern recognition. The model describes the structure of recognition algorithm and parameters necessary for choosing particular algorithm in the model. In the framework of a model, the algorithms differ by their parameters and, therefore, by the way of their classification of the given objects. The results of applying the algorithm of a model to the test sample show the adequacy of this algorithm to the problem at hand. Thus, all of the algorithms of a given model can be supplied with a quality functional.

The choice and/or synthesis of the algorithm, extreme according to the quality functional, presents the main problem in implementing AEC in practice. This problem is closely connected to the reduction of the computational complexity of AEC. The algorithms of acceptable computational complexity are based on efficient formulas which model the algorithm‘s performance; these are formulas for calculating proximity estimates of objects under recognition and precedents. The complexity of formulas for estimate calculations substantially depends on the AEC parameters, such as the system of support sets and the type of proximity function. A recognition algorithm uses a system of support sets as a system of feature subsets for matching object descriptions. A proximity function defines whether the matched objects are ’close’.

Although the efficient formulas are constructed almost for every AEC model of practical interest, the problem of constructing such formulas in the case where the objects are images, object descriptions are 2D matrices, and support sets are spatial (2D) objects still needs to be solved. As was noted above, the AEC class was specialized to operate with 2D object descriptions called a class of algorithms of estimate calculation from two-dimensional information

(2D-AEC). Note that, by now, the efficient formulas are con- structed for one 2D-AEC subclass only: for a subclass with a square as a generative element of the system of support sets.

Here, we propose the method for constructing the efficient algorithms for the 2D-AEC subclass with two-dimensional support sets generated by a rectangle. The idea underlying this procedure consists in transforming the rectangle with the sides $R_1 * R_2$ into the unit square by compressing the plane along the one side R_1 times and along the other R_2 times.

Let us recall the basic objects and properties of the AEC model Generally, a recognition algo- rithm contains a recognizing operator and a decision rule. In AEC, a recognizing operator converts a standard description of object S , subject to recognition into a set of numerical estimates $(\Gamma_1(S), \Gamma_2(S), \ldots, \Gamma_l(S))$, where l is a number of clases. A decision rule helps us to construct the information vector $(\alpha_1, \alpha_2, \ldots, \alpha_l)$, $a_j \in \{0, 1, \Delta\}$, from this set. Here, $\alpha_j = 0$ if an algorithm does not assign object S to jth class; and $\alpha_j = \Delta$ if an algorithm cannot classify object S . To define a recognizing operator, it is necessary to assign a system of support sets, proximity function, feature weights, and precedent weights. Let us consider these parameters in detail.

1. System of support sets.

A system of support sets is a totality of nonempty subsets of the feature set $N = \{1, 2, \ldots, n\}$; the object is described by the values of these features. A system of support sets is denoted by Ω_A. Below, we list the examples of support sets.

1.1. $\Omega_A = 2^N$; i.e., a system of support sets is a class of all (nonempty) subsets of feature set N.

1.2. $\Omega_A = \{\Omega | \Omega \subseteq N, |\Omega| = k\}$, where k is an integer and $1 \leq k \leq n$; i.e., a system of support sets consists of all of the subsets of the set N which have a predefined power k , e.g., $\Omega_A = 1, 2, \ldots, n$ for $k = 1$ and $\Omega_A = \{N\}$ for

$k = n$. The following relation connects the systems of support sets of 1.1 and 1.2:

$$2^N = \bigcup_{k=1}^{n} \{\Omega | \Omega \subseteq N, |\Omega| = k\}.$$

1.3. $\Omega_A = \{\Omega | \Omega \subseteq N, |\Omega| \leq k\}$, where k is an integer and $1 \leq k \leq n$; i.e., Ω_A consists of all of the subsets of the set N which have a power no more than the predefined one.

1.4. $\Omega_A = \{\Omega | \Omega \subseteq N, |\Omega| \subseteq \{k_1, k_2, \ldots, k_u\}\}$, where $k_1, k_2, \ldots, k_u$ are integers and $1 \leq k_i \leq n, i = \overline{1,n}$.

Any support set Ω can be encoded by the binary vector $\tilde{\omega}$ of the length n in the following way: the ith coordinate $\tilde{\omega}$ is equal to one if and only if the ith feature is contained in Ω. The thus-constructed $\tilde{\omega}$ vector is called a characteristic vector of the support set Ω. It is obvious that a set Ω and its characteristic vector $\tilde{\omega}$ are connected by a one-to-one correspondence.

Sometimes, it is convenient to consider a system of support sets as a set of characteristic vectors that encode the support sets of the algorithm. In those cases, we consider Ω_A to be a set of vertices $\{\tilde{\omega}\}$ of n-dimensional Boolean cube E^n.

As usual, we denote the norm (weight) of the binary vector $\|\{\tilde{\omega}\}\|$ equal to the number of its unitary coordinates by $\{\tilde{\omega}\}$. The set of all binary vectors of the weight k is called the kth layer of the Boolean cube and denoted by E_n^k.

We call the Boolean function $f_A(\tilde{\omega})$ a characteristic function of the system of support sets Ω_A if $f_A(\tilde{\omega}) = 1 \Leftrightarrow \in \Omega_A$. Obviously, a system of support sets of the algorithm is unambiguously described by its characteristic function.

The characteristic function of a system of support sets of (1.1)-type vanishes only if all its variables vanish.

For a system of support sets of (1.2)-type, the characteristic function is equal to unity in the whole layer of Boolean cube and only there.

For a system of support sets of (1.3)-type, the characteristic function is equal to unity in the whole first, second, ..., kth layers of Boolean cube and only there.

2. Proximity function.

Let $I(S) = (a_1, a_2, \ldots, a_n)$ be a standard (feature) description of object $S, \Omega = \{i_1, i_2, \ldots, i_k\}$, and let be a characteristic vector Ω. We denote a subdescription of the object S represented in the form $(a_{i_1}, a_{i_2}, \ldots, a_{i_k})$ by the symbols $\tilde{\omega}I(S)$ or $\tilde{\omega}(S)$.

The proximity function $B_{\tilde{\omega}}(S, S')$ depends on $\tilde{\omega}$-subdescriptions of objects S, S' and takes two values: 0 if the objects are not close and 1 otherwise.

Most often, the following proximity functions are considered:

2.1

$$B_{\tilde{\omega}}(S, S') = \begin{cases} 1, \tilde{\omega}S = \tilde{\omega}S' \\ 0, \tilde{\omega}S \neq \tilde{\omega}S' \end{cases} \tag{1.1}$$

2.2. Let the metric (or semimetric) $\rho_i(x, y), i = \bar{1, n}$, be defined on the range of definition of the ith feature. Let $\tilde{\omega}S = (a_{i_1}, a_{i_2}, \ldots, a_{i_k}), \tilde{\omega}S' = (b_{i_1}, b_{i_1}, \ldots, b_{i_1})$, and the quantities $\varepsilon_i \geq 0, i = \bar{1, n}, \varepsilon \geq 0$, where ε is integer, be set. Consider a system of inequalities

$$\rho_{i_1}(a_{i_1}, b_{i_1}) \leq \varepsilon_{i_1}, \rho_{i_2}(a_{i_2}, b_{i_2}) \leq \varepsilon_{i_2}, \ldots, \rho_{i_k}(a_{i_k}, b_{i_k}) \leq \varepsilon_{i_k} \tag{1.2}$$

and denote the number of unsatisfied inequalities in this system by γ. Then,

$$B_{\tilde{\omega}}(S, S') = \begin{cases} 1, \gamma \leq \varepsilon \\ 0, \gamma > \varepsilon \end{cases} \tag{1.3}$$

The parameters of this function are the vector $\varepsilon = (\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n)$ and the quantity ε (maximally admissible number of unsatisfied inequalities in system (1.2)). If $\varepsilon_i = 0, i = \bar{1, n}$, and $\varepsilon = 0$, this proximity function is identically equal to the proximity function determined in 2.1.

2.3. If in the conditions of the previous point, we set two integers ε^1 and ε^2 (ε^1, $\varepsilon^2 \geq 0$) instead of ε, then

$$B_{\tilde{\omega}}(S, S') = \begin{cases} 1, \|\{\tilde{\omega}\}\| - \gamma \geq \varepsilon^1, \gamma \leq \varepsilon^2 \\ 0, \gamma > \text{otherwise} \end{cases} \tag{1.4}$$

Vector $\varepsilon = (\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n)$ and quantities ε^1 and ε^2 (minimal accessible number of satisfied inequalities in system (1.2) and maximal accessible number of unsatisfied inequalities in system (1.2), respectively) are parameters of the function. For $\varepsilon^1 = 0$ and $\varepsilon^2 = \varepsilon$, we get the (2.2)-type proximity function.

Suppose once more that $I(S) = (a_1, a_2, \ldots, a_n)$, $I(S') = (b_1, b_2, \ldots, b_n)$, the metric (or semimetric) $\rho_i(x, y)$ is defined in the range of definition of the ith feature, and the values $\varepsilon_i \geq 0, i = \bar{1, n}$ are set. The binary vector $\tilde{\delta} = \tilde{\delta}(S, S') = (\tilde{\delta}_1, \tilde{\delta}_2, \ldots, \tilde{\delta}_n)$ defined as follows:

$$\delta_i = \begin{cases} 1, \rho_i(a_i, b_i) \leq \varepsilon_i \\ 0, \rho_i(a_i, b_i) > \varepsilon_i \end{cases} \tag{1.5}$$

where $i = \bar{1, n}$, is called a characteristic vector of the proximity of objects S and S'.

By using a characteristic vector of proximity, we can rewrite the expression for (2.2)-type proximity function as

$$B_{\tilde{\omega}}(S, S') = \begin{cases} 1, (\tilde{\delta}', \tilde{\omega}) \leq \varepsilon \\ 0, (\tilde{\delta}', \tilde{\omega}) > \varepsilon \end{cases} \tag{1.6}$$

where $\tilde{\delta}'$ is a binary vector obtained by the coordinate-wise negation of vector $\tilde{\delta}$ and (α, β) is a scalar product of the vectors α and β which is equal to the sum of their coordinatewise multiplications.

In a similar way, for the (2.3)-type proximity function,

$$B_{\tilde{\omega}}(S, S') = \begin{cases} 1, (\tilde{\delta}, \tilde{\omega}) \geq \varepsilon^1, (\tilde{\delta}', \tilde{\omega}) \geq \varepsilon^2 \\ 0, \text{otherwise} \end{cases} \tag{1.7}$$

We can introduce vector $\tilde{\delta}(S, S')$ while ignoring the metric ρ_i and quantities ε_i in the following way:

$$\delta_i = \begin{cases} 1, a_i = b_i \\ 0, a_i \neq b_i \end{cases} \tag{1.8}$$

In this case, the expression for the (2.1)-type proximity function can be rewritten in the following way:

$$B_{\tilde{\omega}}(S, S') = \begin{cases} 1, (\tilde{\delta}, \tilde{\omega}) = \|\tilde{\omega}\| \\ 0, (\tilde{\delta}, \tilde{\omega}) < \|\tilde{\omega}\| \end{cases} \tag{1.9}$$

3. Feature weights.

Feature weights are set by the vector $\mathbf{p} = (p_1, p_2, \ldots, p_n), p_i > 0, i = \bar{1, n}$.

Let $i_1, i_2, \ldots, i_k$ be a set of the indices of all unit coordinates of the characteristic vector $\tilde{\omega}$. The weight of the support set Ω with a characteristic vector $\tilde{\omega}$ is denoted by $p(\tilde{\omega}); p(\tilde{\omega}) = p_{i_1} + p_{i_2} + \ldots + p_{i_k}$.

4. Precedent weights.

Precedent weights are defined by the vector $\gamma = (\gamma_1, \gamma_2, \ldots, \gamma_m)$, where $\gamma_q = \gamma(S_q) > 0, q = \bar{1, m}$, and m is a total number of precedents. This point concludes the list of the parameters of the recognition operator of AEC.

The estimate $\Gamma_j(S)$ of the object S over the jth class is defined by the following formula:

$$\Gamma_i(S) = \frac{1}{K} \frac{1}{|W_j|} \sum_{S' \in W_j} \gamma(S') \sum_{\tilde{\omega} \in \Omega_A} p(\tilde{\omega}) B_{\tilde{\omega}}(S, S'), j = \bar{1, l}, \tag{1.10}$$

where K is a normalized coefficient and W_j is a set of precedents of the jth class.

Sometimes we use the formulas to assess an esti- mate $\Gamma_j(S)$ that differ from Eq. (1.10). In any case, however, the semantics of the initial formulas for

Γ_j is the same; i.e., over all of the support sets, the value of proximity function (and/or of its negation) is calculated for the given object S and for each object S' from the learning set. Each time, the weights of the features and the precedents are equally accounted of.

We finish the definition of recognition algorithm by setting the decision rule (see [13, 16]).

Note one important property of the estimate (1.10). The following equality is true:

$$\Gamma_j^S(\Omega_A^1 \cup \Omega_A^2) = \Gamma_j^S(\Omega_A^1) + \Gamma_j^S(\Omega_A^2) - \Gamma_j^S(\Omega_A^1 \cap \Omega_A^2),$$

where $\Gamma_j^S(\Omega_A^t$ is an estimate $\Gamma_j(S)$, defined according the system of support sets $\Omega_A^t, t = 1, 2$. This equality admits the generalization for the case of a union of any finite number of the support set systems.

The estimate $\Gamma_j^S \bigcup\limits_{t=1}^{u} \Omega_A^t$ over the union of mutually disjoint systems of support sets is merely the sum of estimates $\Gamma_j^S(\Omega_A^t$ over all $t = \overline{1,u}$. Therefore, the estimate $\Gamma_j(S)$ is additive with respect to the union of disjoint systems of the support sets.

This property allows us to easily obtain the estimate $\Gamma_j^S \bigcup\limits_{t=1}^{u} \Omega_A^t$ if the estimates $t = \overline{1,u}$ if the estimates $\Gamma_j^S(\Omega_A^t)(\Omega_A^{t_1} \cap \Omega_A^{t_2} \neq \emptyset, t_1 \neq t_2)$ are known. Suppose, for example, that Ω_{2^N} is a (1.1)-type system of support sets and Ω_k is (1.2)-type system of support sets with a given power k of N subsets. Then,

$$\Gamma_j^S(\Omega_{A^N}) = \sum_{k=1}^{n} \Gamma_j^S(\Omega_k).$$

Hereinafter, for the convenience, we omit the multiplier $\frac{1}{K}\frac{1}{|W_j|}$ in Eq. (1.10):

$$\Gamma_j(S) = \sum_{S' \in W_j} \gamma(S') \sum_{\tilde{\omega} \in \Omega_A} p(\tilde{\omega}) B_{\tilde{\omega}}(S, S'), j = \overline{1,l} \qquad (1.11)$$

Overview of phylogenetic tree reconstruction stages. There are several methods for reconstructing phylogenetic trees. But, in the thesis we will discus about distance-based methods.

Construction of a phylogenetic tree can be divided into three major steps.

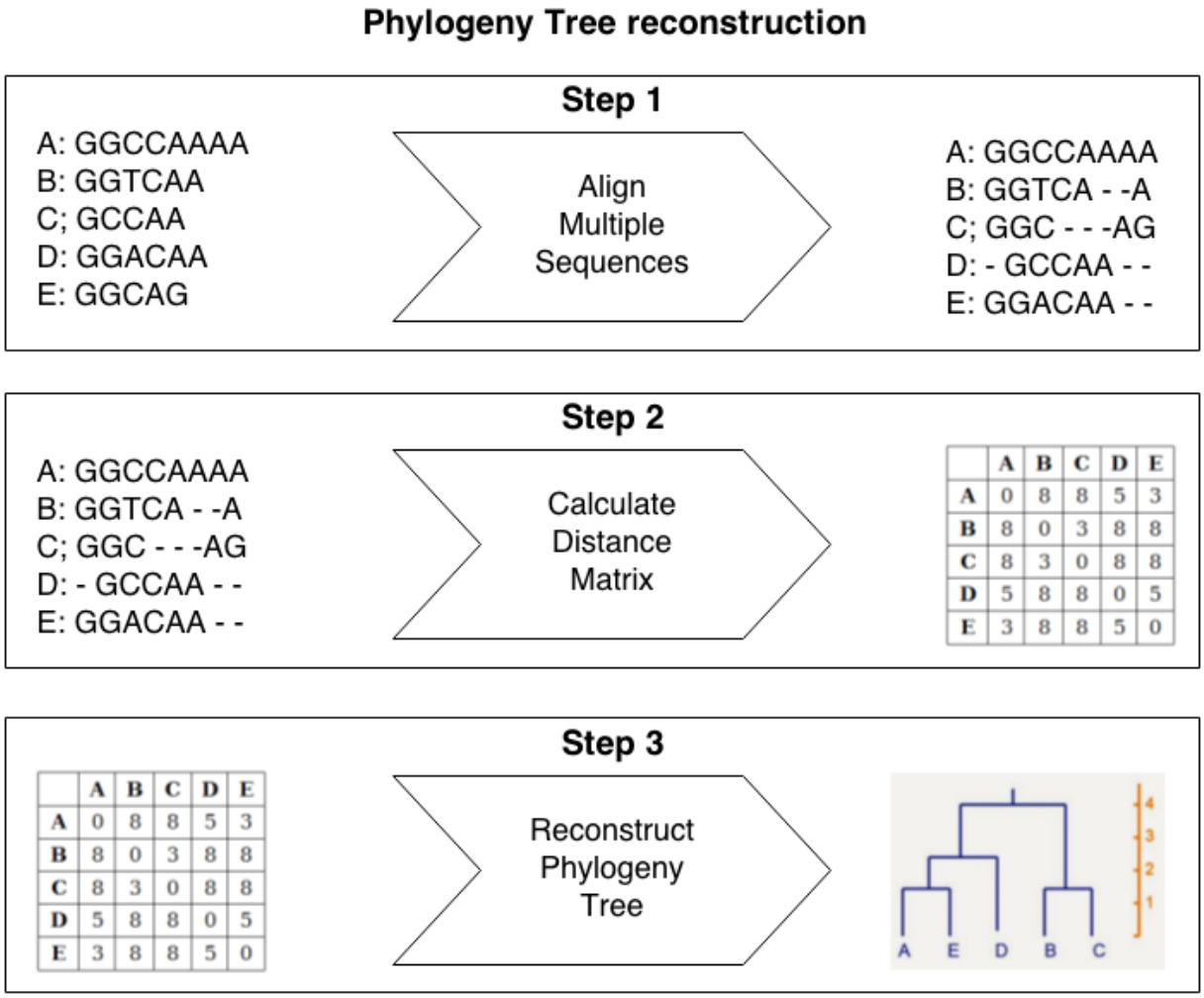

Figure 1.6. Main steps to reconstruct phylogenetic tree.

The next chapter will be devoted to these steps.

Conclusion for Chapter 1

Conclusions made in this chapter:

- Defined goals and objectives based on the data in this chapter.

- Algorithms of estimate calculations can be used to analyze phylogenetic data Figure 1.6.

- Phylogenetic tree reconstruction can be divided to 3 main stages.

- Phylogenetics play an important role in biology
- Main goal of the phylogeny is to reconstruct phylogenetic tree.
- Phylogenetic tree represents evolutionary histories of living organisms.

Chapter 2. The main stages of the construction of a phylogenetic tree

1. Multiple sequences alignment

To show the novelty of my work, at first need to understand what was researched previously.

Same stages of the actions are used to construct a phylogenetic tree in many methods. They can be divided in three main actions.

Stage 1: Align sequences.

Data: Given n unaligned sequences;

Result: n aligned sequences;

if *All sequnces have same length* **then**

    go to next phase;

else

    Align all sequences;

Alignment methods vary in type of data [8] (DNA, RNA, or amino-acid) they can handle, and also, to some extent, the objectives of the alignment method [9]. Thus, some methods are designed exclusively for proteins, some exclusively for RNAs, but many alignment methods can analyze both protein and nucleotide datasets. We refer to methods that can analyze all types of molecular sequences as "generic" methods.

Sequence alignment is a way of arranging sequences of DNA, RNA, or proteins in order to distinguish regions of similarity. A multiple sequence alignment (MSA) is a sequence alignment of three or more biological sequences such as protein, DNA, or RNA. Typically it is implied that the set of sequences share an evolutionary relationship, which means they are all descendents from a common ancestor. These regions may correspond to functional, structural, or evolutionary relationships between the sequences. Alignments can reflect a

degree of evolutionary change between sequences that are descendants from a common ancestor. There is a relationship between phylogenies and sequence alignments.

To find the globally optimum alignment, a dynamic programming technique can be used if one uses a parsimony approach and a particular scoring scheme. There is no universally agreed upon scoring scheme. This approach is computationally expensive and impractical since it has been shown to be a NP-complete problem. Instead, heuristics are commonly used to perform a multiple sequence alignment. This research focused on studying one heuristic approach called progressive alignment. One popular program that employs a progressive alignment method is ClustalW [1]. Perform a multiple sequence alignment can be broken down into three major steps.

FIRST , all pairs of sequences are aligned separately and then a distance matrix is calculated giving the divergence of each pair of sequences. A full dynamic programming alignment is calculated for each pair using two gap penalties, one for opening a gap and another for extending a gap. The score in the distance matrix is computed by taking the number of identities in the best alignment divided by the number of residues compared excluding gap positions. Then that number is multiplied by 100 and subtracted from 1.0 to give a value between 0 and 1.0.

SECOND , a guide tree is calculated which will be used to guide the final multiple alignment process. This tree is calculated by using the distance matrix from the first step and a Neighbor-Joining clustering algorithm. Weights are also assigned to each sequence depending on their distance from the root of the tree. By contrast, in the original Clustal progressive alignment algorithm, all sequences would be equally weighted.

[1] ClustalW is a popular program used for multiple sequence alignment and for preparing phylogenetic trees

THIRTD , the sequences are progressively aligned according to the branching order in the guide tree. To do this a series of pairwise alignments are used to align larger and larger groups of sequences. First, proceed from the tips of the rooted tree towards the root. At each alignment a full dynamic programming algorithm is used with penalties for opening and extending gaps. Each step aligns two existing alignments or sequences. Gaps that are present in the older alignments stay in place. When all the sequences have been considered a final alignment is produced. That final alignment can then be used to construct a phylogenetic tree for those species.

Disadvantages: One disadvantage of a progressive alignment approach is that, once an alignment has been performed involving some of the species, this alignment is never reconsidered despite what other decisions are made for the remaining species. This can lead to inaccuracies in the final alignment.

2. Building Distance Matrix to campare taxons

Stage 2: Building Distance Matrix. An important tool in distance-based methods in building phylogenetic tree is the use of distance matrix. An $m * n$ matrix is a collection of nm real numbers, arranged in m rows and n columns.

$$M = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}$$

Each number a_{ij} in the matrix is called an $entry$ of the $matrix$. A matrix is said to be $symmetric$ if for any row and column $i, j, a_{ij} = a_{ji}$. The following examples show the symmetry and asymmetry of matrices.

In particular, if a matrix has the same number of rows and columns, $n*n$, it is often called a square matrix. Given a collection of n sequences, with distance d defined between any pair of sequences, the following matrix

$$M = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{bmatrix}$$

is call the distance matrix, where d_{ij} is the distance between the ith and jth sequences.

One of chanllenges in using distance matrices with distance methods to build phylogenetic tree is the building of the matrix. The very necessary condition to make all the thory work is that the distance must satisfy metric property. That is, all the entries in the distance matrix representing distances between sequences must satisfy the triangulr inequality,

$$d_{ik} \leq d_{ij} + d_{jk},$$

for all $1 \leq i, j, k \leq n$. Unfortunately, many times it is not the case in practice. There are always some entries fail the inequality, if not all of them.

Our first algorithm is to correct any possible errors to make the matrix symmetric. Assuming that, upon detection of asymmetries, there is no way algorithm would know which one is the true correct data, it simply uses the

average of the two values to substitute. The pseudo-code for the algorithm is given in the following.

Algorithm: Symmetry Enforcing - Iteration

Data: n is the number of nodes, i=1;

Result: retrieve data matrix in a double string;

while *i!=n* **do**

    j=2;

    if *d[ij]!=d[ji]* **then**

        substitute d[ij] and d[ji] by (d[ij]+d[ji])/2;

    j=j+1;

    if *j=n* **then**

        i=i+1;

The complexity of the algorithm is linear since the matrix has $O(n^2)$ entries.

Metric distance matrix. Assume we now have a symmetric distance matrix, our next challenge is to enforce the metric property. Again, our assumption is that the algorithm would not know the true correct data, it simply uses the minimum value to make the distance metric. More specifically, for a node A_n, if the triangular inequality holds, then for any $j, k < i$, we must have

$$d_{ik} \geq d_{ij} - d_{jk},$$

Thus,

$$d_{ik} \geq \max_{k<j}(d_{ij} - d_{jk}),$$

is a necessary condition for the triangular inequality to be true. The pseudo-code for the algorithm is given in the following.

Algorithm: Metricalization of distance matrix

Data: n is the number of nodes; i=3;

Result: retrieve data matrix M[n] in a double string;

while *i!=n+1* **do**

    j=2;

    k=1;

    if *d[ij]¡d[jk]-d[ik]* **then**

        substitute d[ij] by d[jk]-d[ik];

    k=k+1;

    if *k=j* **then**

        j=j+1;

    if *j=i* **then**

        i=i+1;

The complexity of the algorithm will also be polynomial since

$$\sum_{i}^{n} i * i(i+1) \cong O(n^4)$$

Also, the correction process is heavily based on the earlier portion of the data than the later portion. Namely, if a distance matrix fails the triangular property only at the last node or so, the correction is very minimum. On the other hand, if it occurs at the first three (automatically true with only two nodes) nodes, then it might have impact on all the data follows.

3. Phylogenetic tree generation

Stage 3: Reconstructing phylogenetic tree. Before proceeding to the construction of a phylogenetic tree, you should examine the existing models of gene sequences.

Stochastic models of sequence evolution. We begin with a description of the simplest stochastic models of DNA sequence evolution, and then discuss amino-acid sequence evolution models and codon evolution models. The simplest models of DNA sequence evolution treat the sites within the sequences independently. Thus, a model of DNA sequence evolution must describe the probability distribution of the four states, A, C, T, G, at the root, the evolution of a random site (i.e., position within the DNA sequence) and how the evolution differs across the sites. Typically the probability distribution at the root is uniform (so that all sequences of a fixed length are equally likely). The evolution of a single site is modeled through the use of "stochastic substitution matrices", $4x4$ matrices (one for each tree edge) in which every row sums to 1. A stochastic model of how a single site evolves can thus have up to 12 free parameters. The simplest such model is the Jukes-Cantor model, with one free parameter, and the most complex is the General Markov model, with all 12 parameters:

Definition 1. *The General Markov (GM) model of single-site evolution is defined as follows.*

1. *The nucleotide in a random site at the root is drawn from a known distribution, in which each nucleotide has positive probability.*

2. *The probability of each site substitution on an edge e of the tree is given by a $4x4$ stochastic substitution matrix $M(e)$ in which $det(M(e))$ is not 0, 1, or -1.*

This model is generally used in a context where all sites evolve identically and indpendently (the i.i.d. assumption), with rates of evolution drawn typically drawn from a gamma distribution. (Note that the distribution of the rates-across-sites has an impact on phylogeny estimation and dating, as discussed by Evans and Warnow.) In what follows, we will address the simplest version of the GM model so that all sites have the same rate of evolution.

We denote a model tree in the GM model as a pair, $(T, M(e) : e \in E(T))$, or more simply as (T, M). For each edge $e \in E(T)$, we define the length of the edge $\lambda(e)$ to be - $log|det(M(e))|$. This allows us to define the matrix of leaf-to-leaf distances, λ_{ij}, where $\lambda_{ij} = \sum_{e \in P_{ij}} \lambda(e)$, and where P_{ij} is the path in T between leaves i and j. A matrix defined by path distances in a tree with edge weights is called "additive", and it is a well-known fact that given any additive matrix, it is easy to recover the underlying leaf-labelled tree T for that matrix in polynomial time.

This general model of site evolution subsumes the great majority of other models examined in the phylogenetic literature, including the popular General Time Reversible (GTR) model, which requires only that $M(e) = M(e')$ for all edges e and e'. Further constraints on the matrix $M(e)$ produce the Hasegawa-Kishino-Yang (HKY) model, the Kimura 2-parameter model (K2P), the Kimura 3-ST model (K3ST), the Jukes-Cantor model (JC), etc. These models are all special cases of the General Markov model, because they place restrictions on the form of the stochastic substitution matrices. The standard model used for nucleotide phylogeny estimation is GTR+gamma, i.e., the General Time Reversible (GTR) model of site substitution, equipped with a gamma distribution of rates across sites.

Protein models. Just as with DNA sequence evolution models, there are Markov models of evolution for amino-acid sequences, and also for coding DNA sequences. These models are described in the same way - a substitution matrix that governs the tree, and then branch lengths. While the GTR model can be extended to amino-acids (to produce a $20x20$ matrix) or to codon-based models (to produce a $64x64$ matrix), both of which must be estimated from the data, in practice these models use fixed matrices, each of which was estimated from external biological data. The most well known protein model is the Dayhoff model, but improved models have been developed in recent years.

Similarly, codon-based models have also been based on fixed $64x64$ matrices. In practice, the selection of a protein model for a given dataset is often done using a statistical test, such as ProtTest, and then fixed. In the subsequent tree estimation performed under that model, only the tree and its branch lengths need to be estimated.

Phylogeny Estimation Methods. There are many different phylogeny estimation methods, too numerous to mention here. However, the major ones can be classified into the following types:

- distance-based methods, which first compute a pairwise distance matrix (usually based on a statistical model) and then compute the tree from the matrix,
- maximum parsimony and its variants, which seek a tree with a total minimum number of changes (as defined by edit distances between sequences at the endpoints on the edges of the tree),
- maximum likelihood, which seeks the model tree that optimizes likelihood under the given Markov model, and
- Bayesian MCMC methods, which return a distribution on trees rather than a single tree, and also use likelihood to evaluate a model tree.

Statistical Performance Criteria. We discuss three concepts here: *identifiability, statistical consistency, and sequence length requirements*.

Identifiability: A statistical model or one of its parameters is said to be "identifiable" if it is uniquely determined by the probability distribution defined by the model. Thus, in the context of phylogeny estimation, the unrooted model tree topology is identifiable if it is determined by the probability distribution (defined by the model tree, which includes the numeric parameters) on the patterns of nucleotides at the leaves of the tree. In the case of nucleotide models,

the state at each leaf can be A,C,T, or G, and so there are 4^n possible patterns in a tree with n leaves (similarly, there are 20^n possible patterns for amino-acid models). It is well known that the unrooted tree topology is identifiable under the General Markov model, and recent work has extended this to other models.

Statistical Consistency: We say that a method Φ is "statistically consistent" for estimating the topology of the model tree (T, θ) if the trees estimated by Φ converges to the unrooted version of T (denoted by T^u) as the number of sites increases. (Note that under this definition, we are not concerned with estimating the numeric parameters.) Equivalently, for all $\varepsilon > 0$ there is a sequence length K so that if a set S of sequences of length $k \geq K$ are generated by (T, θ), then the probability that $\Phi(S) = T^u$ is at least $1 - \varepsilon$. We say that a method is statistically consistent under the GM model if it is statistically consistent for all model trees in the GM model. Similarly, we say a method is statistically consistent under the GTR model if it is statistically consistent under all model trees in the GTR model.

Many phylogenetic methods are statistically consistent under the GM model, and hence also under its submodels (e.g., the GTR model). For example, maximum likelihood, neighbor joining (and other distance-based methods) for properly computed pairwise "distances", and Bayesian MCMC methods, are all statistically consistent. Onthe other hand, maximum parsimony and maximum compatibility are not statistically consistent under the GM model. In addition, it is well known that maximum likelihood can be inconsistent if the generative model is different from the model assumed by maximum likelihood, but maximum likelihood can even be inconsistent when its assumptions match the generative model, if the generative model is too complex! For example, Tuffey and Steel showed that maximum likelihood is equivalent to maximum parsimony under a very general "no-common-mechanism" model, and so is inconsistent under this model. In this case, the model itself is not identifiable, and this

is why maximum likelihood is not consistent. However, there are identifiable models for which ML is not consistent, as observed by Steel.

Sequence length requirement: Clearly, statistical consistency under a model is a desirable property. However, statistical consistency does not address how well a method will work on finite data. Here, we address the "sequence length requirement" of a phylogeny estimation method Φ, which is the number of sites that Φ needs to return the (unrooted version of the) true tree with probability at least $1 - \epsilon$ given sequences that evolve down a given model tree (T, θ). Clearly, the number of sites that suffces for accuracy with probability at least $1 - \epsilon$ will depend on Φ and ϵ, but it also depends on both T and θ.

We describe this concept in terms of the Jukes-Cantor model, since this is the simplest of the DNA sequence evolution models, and the ideas are easiest to understand for this model. However, the same concepts can be applied to the more general models, and the theoretical results that have been established regarding sequence length requirements extend to the GM (General Markov) model, which contains the GTR model and all its submodels.

In the Jukes-Cantor (JC) model, all substitutions are equally likely, and all nucleotides have equal probability for the root state. Thus, a Jukes-Cantor model tree is completely defined by the rooted tree T and the branch lengths $\lambda(e)$, where $\lambda(e)$ is the expected number of changes for a random site on the edge e. It is intuitively obvious that as the minimum branch length shrinks, the number of sites that are needed to reconstruct the tree will grow, since a branch on which no changes occur cannot be recovered with high probability (the branch will appear in an estimated tree with probability at most one-third, since at best it can result from a random resolution of a node of degree at least 4). It is also intuitively obvious that as the maximum branch length increases, the number of sites that are needed will increase, since the two sides of the long branch will seem random with respect to each other. Thus, the sequence length requirement for a given method to be accurate with probability at least $1-\epsilon$ will

be impacted by the shortest branch length f and the longest branch length g. It is also intuitively obvious that the sequence length requirement will depend on the number of taxa in the tree.

Expressing the sequence length requirement for the method Φ as a function of these parameters (f, g, n and ϵ) enables a different - and finer - evaluation of the method's performance guarantees under the statistical model. Hence, we consider f, g, and ϵ as fixed but arbitrary, and we let JCf,g denote all Jukes-Cantor model trees with $0 < f \leq \lambda(e) \leq g < \infty$ for all edges e. This lets us bound the sequence length requirement of a method as a function only of n, the number of leaves in the tree.

The definition of "absolute fast convergence" under the Jukes-Cantor model is formu- lated as an upper bound on the sequence length requirement, as follows:

Definition 2. *A phylogenetic reconstruction method Φ is absolute fast-converging (afc) for the Jukes-Cantor (JC) model if, for all positive f, g, and ε, there is a polynomial $p(n)$ such that, for all (T, θ) in $JC_{f,g}$, on set S of n sequences of length at least $p(n)$ generated on T, we have $Pr[\Phi(S) = T^u] > 1 - \varepsilon$.*

Note also that this statement only refers to the estimation of the unrooted tree topology T^u and not the numeric parameters Φ. Also, note that the method Φ operates without any knowledge of parameters f or g-or indeed any function of f and g. Thus, although the polynomial p depends upon both f and g, the method itself will not. Finally, this is an upper bound on the sequence length requirement, and the actual sequence length requirement could be much lower.

The function $p(n)$ can be replaced by a function $f(n)$ that is not polynomial to provide an upper bound on the sequence length requirement for methods that are not proven to be absolute fast converging.

Empirical Performance. So far, these discussions have focused on theoretical guarantees under a model, and have addressed whether a method will converge to the true tree given long enough sequences (i.e., statistical consistency), and if so, then how long the sequences need to be (sequence length requirements). However, these issues are purely theoretical, and do not address how accurate the trees estimated by methods are in practice (i.e., on data). In addition, the computational performance (time and memory usage) of phylogeny estimation methods is also important, since a method that is highly accurate but will use several years of compute time will not generally be useful in most analyses.

Phylogenetic tree accuracy can be computed in various ways, and there are substantive debates on the "right" way to calculate accuracy; however, although disputed, the Robinson-Foulds (RF) distance, also called the "bipartition distance", is the most commonly used metric on phylogenetic trees. We describe this metric here.

Given a phylogenetic tree T on n taxa, each edge can be associated with the bipartition it induces on the leaf set; hence, the tree itself can be identified with the set of leaf- bipartitions defined by the edges in the tree. Therefore, two trees on the same set of taxa can be compared with respect to their bipartition sets. The RF distance between two trees is the size of the symmetric difference of these two sets, i.e., it is the number of bipartitions that are in one tree's dataset but not both. This number can be divided by $2(n-3)$ (where n is the number of taxa) to obtain the "RF rate". In the context of evaluating phylogeny estimation methods, the RF distance is sometimes divided into false negatives and false positives, where the false negatives (also called "missing branches") are branches in the true tree that are not present in the estimated tree, and the false positives are the branches in the estimated tree that are not present in the true tree. This distinction between false positives and false negatives enables a more detailed comparison between trees that are not binary.

Many studies have evaluated phylogeny estimation methods on simulated data, varying the rate of evolution, the branch lengths, the number of sites, etc. These studies have been enormously informative about the differences between methods, and have helped biologists make informed decisions regarding methods for their phylogenetic analyses. Some of the early simulation studies explored performance on very small trees, including the fairly exhaustive study by Huelsenbeck and Hillis on 4-leaf trees, but studies since then have explored larger datasets and more complex questions. For example, studies have explored the impact of taxon sampling on phylogenetic inference, the impact of missing data on phylogenetic inference, and the number of sites needed for accuracy with high probability. In fact, simulation studies have become, perhaps, the main way to explore phylogenetic estimation.

Distance-based methods. Distance-based methods operate by first computing a matrix of distances (typically using a statistically defined technique, to correct for unseen changes) between every pair of sequences, and then construct the tree based on this matrix. Most, but not all, distance-based methods are statistically consistent, and so will be correct with high probability, given long enough sequences. In general, distance-based methods are polynomial time, and so have been popular for large-scale phylogeny estimation. While the best known distance-based method is probably neighbor joining, there are many others, and many are faster and/or more accurate.

One of the interesting properties about distance-based methods is that although they are typically guaranteed to be statistically consistent, not all distance-based methods have good empirical performance! A prime example of this lesson is the Naive Quartet Method, a method that estimates a tree for every set of four leaves using the Four-Point Method (a statistically-consistent distance method) and then returns the tree that is consistent with all the quartets if it exists. It is easy to show that the Naive Quartet Method runs in polynomial

time and is statistically consistent under the General Markov model; however, because it requires that every quartet be accurately estimated, it has terrible empirical performance! Thus, while statistical consistency is desirable, in many cases statistically inconsistent methods can outperform consistent ones.

Maximum parsimony. Maximum parsimony (MP) is NP-hard, and so the methods for MP use heuristics (most without any performance guarantees). The most efficient and accurate maximum parsimony software for very large datasets is probably Tree analysis using New Technology (TNT), but PAUP* is also popular and effective on datasets that are not extremely large. TNT is a particularly effective parsimony heuristic for large trees, and has been able to analyze a multi-marker sequence dataset with more than 73,000 sequences.

Maximum likelihood. Maximum likelihood (ML) is also NP-hard, and so attempts to solve ML are also made using heuristics. While the heuristics for MP used to be computationally more efficient than the heuristics for ML, the current set of methods for ML are quite effective at "solving" large datasets. (Here the quotes indicate that there is no guarantee, but reasonably good results do seem to be obtained using the current best software.)

The leading methods for large-scale ML estimation under the General Time-Reversible (GTR) + Gamma model include Randomized Axelerated Maximum Likelihood (RAxML)[2], FastTree-2, PhyML, and Genetic Algorithm for Rapid Likelihood Inference (GARLI) [3]. Of these four methods, RAxML is clearly the most frequently used ML method, in part because of its excellent parallel implementations. However, a recent study showed that trees estimated by FastTree-2 were almost as accurate as those estimated by RAxML, and that FastTree-2 finished in a fraction of the time; for example, FastTree-2 was able to analyze an alignment with almost 28,000 ribosomal ribonucleic acid (rRNA) sequences in about 5 hours, but RAxML took much longer. Fur-

[2]The Exelixis Lab's standard tool for Maximum- likelihood based phylogenetic inference

[3]A program for inferring phylogenetic trees

thermore, FastTree-2 has been used to analyze larger datasets (ones with more sequences) than RAxML: the largest dataset published with a RAxML analysis had 55,000 nucleotide sequences, but FastTree has analyzed larger datasets. For example, FastTree-2 has analyzed a dataset with more than 1 million nucleotide sequences, and another with 330,556 sequences. The reported running time for these analyses are 203 hours for the million-taxon dataset, and 13 hours (with 4 threads) for the 330K- taxon dataset [4]. By comparison, the RAxML analysis of 55,000 nucleotide sequences took between 100,000 and 300,000 CPU hours [5]. The difference in running time is substantial, but we should note two things: the RAxML analysis was a multi-marker analysis, and so the sequences were much longer (which impacts running time), and because RAxML is highly parallelized, the impact of the increased running time is not as significant (if one has enough processors). Nevertheless, for maximum likelihood analysis of alignments with large numbers of sequences, FastTree-2 provides distinct speed advantages over RAxML.

There are a few important limitations for FastTree-2, compared to RAxML. First, FastTree-2 obtains its speed by somewhat reducing the accuracy of the search; thus, the trees returned by FastTree-2 may not produce maximum likelihood scores that are quite as good as those produced by RAxML. Second, FastTree-2 doesn't handle very long alignments with hundreds of thousands of sites very well, while RAxML has a new implementation that is designed specifically for long alignments. Third, FastTree-2 has a smaller set of models for amino-acid analyses than RAxML. Therefore, in some cases (e.g., for wide alignments, and perhaps for amino-acid alignments), RAxML may be the preferred method.

However, the ML methods discussed above estimate trees under the GTR+Gamma model, which has simplifying assumptions that are known to

[4] Morgan Price, personal communication, May 1, 2013.

[5] Alexis Stamatakis, personal communication, May 1, 2013.

be violated in biological data. The nhPhyml method is a maximum likelihood method for estimating trees under the non-stationary, non-homogeneous model of Galtier and Guoy, and hence provides an analytical advantage in that it can be robust to some violations of the GTR+Gamma model assumptions. However, nhPhyml seems to be able to give reliably good analyses only on relatively small datasets (i.e., with at most a few hundred sequences, or fewer sequences if they are very long). The explanation is computational - it uses NNI (nearest neighbor interchanges, see below) to search treespace, but NNI is relatively ineffective, which means that it is likely to get stuck in local optima. This is unfortunate, since large datasets spanning substantial evolutionary distances are most likely to exhibit an increased incidence in model violations. Therefore, highly accurate phylogeny estimation of large datasets may require the use of new methods that are based upon more realistic, and more general, models of sequence evolution [11].

Comparisons between methods. [12] Simulation studies have shown some interesting differences between methods. For example, the comparison between neighbor joining and maximum parsimony reveals that the relative performance may depend on the number of taxa and the rate of evolution, with maximum parsimony sometimes performing better on large trees with high rates of evolution, even though the reverse generally holds for smaller trees.

More generally, most simulation studies have shown that maximum likelihood and Bayesian methods (when they can be run properly) outperform maximum parsimony and distance-based methods in many biologically realistic conditions (see Wang et al. for one such study) [14].

Heuristics for exploring treespace. Since both maximum likelihood and maximum parsimony are NP-hard, methods for "solving" these problems use heuristics to explore the space of different tree topologies. These heuristics differ by the techniques they use to score a candidate tree (with the best ones

	SP	MP	UMP	MSN	SD	NJ	NN	SHB	MED	RMD
1	1	1	1	1	1	1	1	1	1	1
	(1000)	(1000)	(1000)	(1000)	(1000)	(1000)	(1000)	**(999)**	**(999)**	(1000)
2	1	1	1	1	1	1	1	1	1	1
	(1000)	(1000)	(1000)	(1000)	(1000)	(1000)	(1000)	(1000)	(1000)	(1000)
3	1	1	1	1	1	1	1	1	1	1
	(1000)	(1000)	(1000)	(1000)	(1000)	(1000)	(1000)	(1000)	(1000)	(1000)
4	1	1	1	1	1	1	1	1	1	1
	(1000)	(1000)	(1000)	(1000)	(1000)	(1000)	**(967)**	(1000)	(1000)	(1000)
5	1	1	1	3	4	1	4	1	1	1
	(1000)	(1000)	(1000)	(1000)	**(998)**	(1000)	**(871)**	(1000)	**(996)**	(1000)
6	1	1	1	9	16	1	129	1	1	1
	(1000)	(1000)	(1000)	(1000)	**(989)**	(1000)	**(666)**	(1000)	**(985)**	(1000)
7	1	1	1	1	1	1	1	1	1	1
	(1000)	(1000)	(1000)	(1000)	(1000)	(1000)	(1000)	(1000)	(1000)	(1000)
8	1	1	1	1	1	1	1	1	1	1
	(1000)	(1000)	(1000)	(1000)	(1000)	(1000)	(1000)	(1000)	(1000)	(1000)
9	1	1	1	1	1	1	1	1	1	1
	(1000)	(1000)	(1000)	(1000)	(1000)	(1000)	(1000)	(1000)	(1000)	(1000)
10	1	1	1	3	4	1	4	1	1	1
	(1000)	(1000)	(1000)	(1000)	(1000)	(1000)	**(895)**	(1000)	(1000)	(1000)
11	1	1	1	8	16	1	169	4	4	1
	(1000)	(1000)	(1000)	(1000)	**(985)**	(1000)	**(638)**	**(999)**	**(975)**	(1000)
12	1	1	1	48	244	1	*	14	41	1
	(1000)	(1000)	(1000)	**(988)**	**(917)**	(1000)	**(327)**	**(997)**	**(889)**	**(998)**
13	1	1	1	8	52	1	30861	2	36	4
	(1000)	(1000)	**(997)**	(1000)	**(938)**	(1000)	**(509)**	(1000)	**(842)**	**(993)**
14	1	2	4	4	2183	1	*	9	*	16
	(1000)	(1000)	**(975)**	(1000)	**(775)**	(1000)	**(182)**	**(999)**	**(391)**	**(956)**
15	1	3	72	4	1308	1	*	227	*	1456
	(1000)	(1000)	**(825)**	(1000)	**(821)**	(1000)	**(8)**	**(983)**	**(18)**	**(751)**
16	3	2	4	9	208	1	*	6	3562.5	4
	(1000)	(1000)	**(984)**	(1000)	**(901)**	(1000)	**(76)**	(1000)	**(623)**	**(981)**
17	3	2	9	8	1369.5	1	*	16	*	16
	(1000)	(1000)	**(933)**	**(999)**	**(816)**	(1000)	**(24)**	(1000)	**(286)**	**(941)**
18	4	4	201	8	624	1	*	206	*	1008.5
	(1000)	(1000)	**(781)**	**(999)**	**(843)**	(1000)	**(0)**	**(969)**	**(20)**	**(770)**

Those in **bold** are combinations in which one or more data set contained more than 5,000,000 trees.
*Less than 50% of inferred networks contained less than 5,000,000 so the median NTR cannot be determined.
doi:10.1371/journal.pone.0001913.t002

Figure 2.1. A comparison of phylogenetic methods [13]

typically using information from previous trees that have already been scored), and how they move within treespace.

Conclusion for Chapter 2

- Alignment methods vary in type of data [8] (DNA, RNA, or aminoacid) they can handle

- There are three main stages to reconstruct phylogenetic tree: sequences aligment, calculate distance matrix, reconstruction tree.

- Neighbor joining is the simplest methods for reconstructing phylogenetic trees.

- Maximum parsimony sometimes performing better on large trees with high rates of evolution, even though the reverse generally holds for smaller trees.

Chapter 3. Design and Implementation PhylogenyUz

In this chapter, discussed about PhylogenyUz, which is phylogenetic analysis system and considered to be a solution to the challenge. Also, discussed about used technologies, system advantages and disadvantages. Given User-Manual for user and scientists and instructions for developers.

1. Developing software to analyze taxons

To solve given problem we have to develop software, which uses algorithms of estimate calculations to reconstruct phylogenetic tree.

It has to be user-friendly, efficient, responsive and cross platform. Also, it is multilingual, can be used English and Uzbek languages in UI.

To achieve this we use Java programming language. Because, Java needs only Java Virtual Machine to run. More about system requirements given in the next sections.

Main idea is to integrate algorithms of estimate calculations and existing methods of reconstruction. For that, used biojava3 [15] library, which is open-source library, written by developers in bioinformatics and has many implementations of existing methods.

We name our software PhylogenyUz. Below given diagram (Figure 3.1) of major actions what user can do in PhylogenyUz.

PhyloWidget [16] used as a framework for the software which is developed by Jordan and Piel.

2. Design and overview of system components

The purpose of this chapter is to present a detailed description of the PhylowidgetUz. It will explain the purpose and features of the system, the interfaces of the system, what the system will do, the constraints under which it must operate and how the system will react to external stimuli. This document is intended

for both the users and the developers of the system and will be proposed to the biology scientists for its approval.

You can see below (Figure 3.1) the overall architecture of the program.

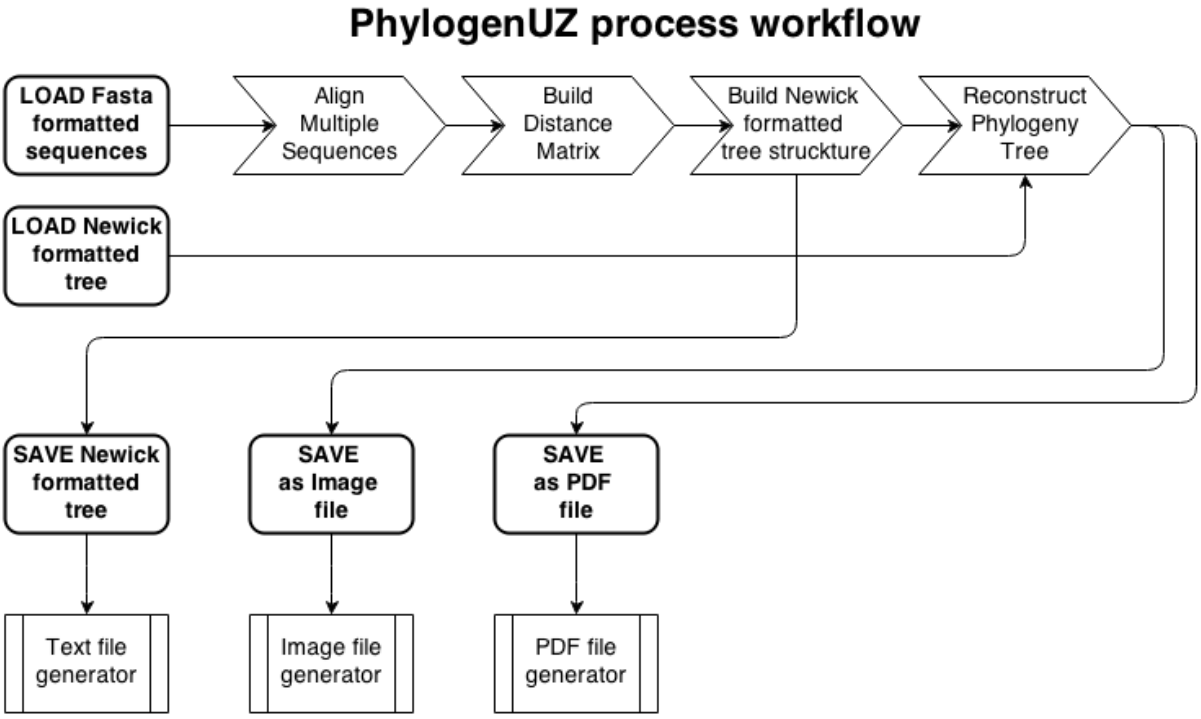

Figure 3.1. Process workflow for reconstructing phylogenetic tree in PhylogenUz

Installation and Run PhylogenyUz

No need to install the software. To start you need to just run in the console command:

java -jar PhylogenUz.jar.

This can cause discomfort for the user who is used to run the program by clicking in the windows operating system.

To solve this problem in the next releases, we want to create separate versions for linux and for windows operation systems.

General features of PhylogenyUz.

Creating New tree

By default, PhylogenyUz creates a new phylogenetic tree on startup. If you want to create a new tree, you need to select New Tree item menu from context menu File or use Ctrl+N shortcuts (Figure 3.2). Then, PhylogenyUz shows basic node on main view.

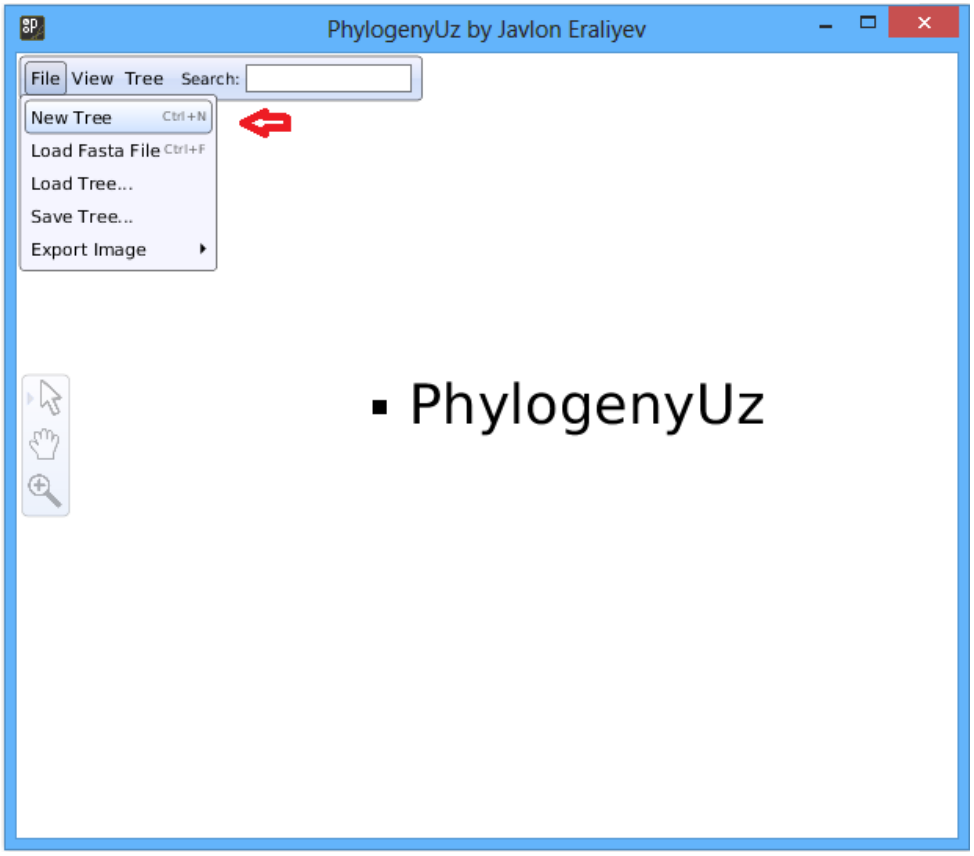

Figure 3.2. Creating new phylogenetic tree in PhylogenUz

Loading Fasta formated files

By default, PhylogenyUz reads Fasta [17] formatted phylogeny sequences. To do that, one needs to select Load Fasta file item menu from context menu File or use Ctrl+F shortcuts.

As soon as, reading completed, PhylogenyUz alignes all sequences, calculates distance matrix, generates Newick formatted tree and shows on main view.

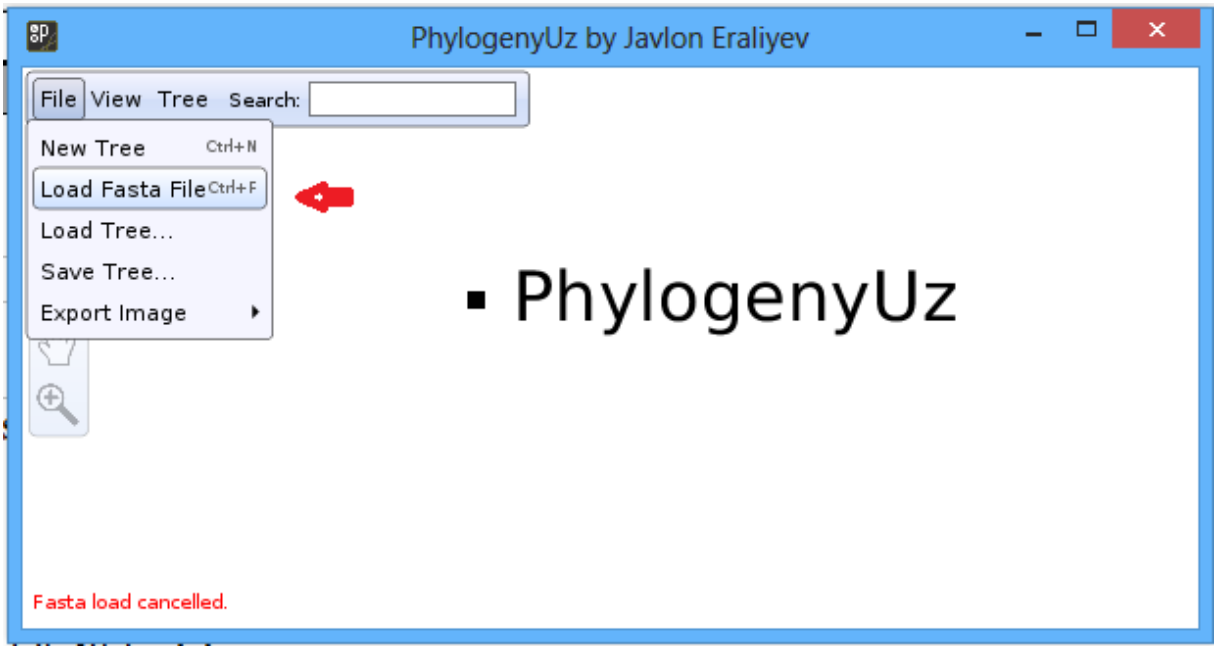

Figure 3.3. Loading fasta formatted files in PhylogenUz

Loading Phylogenetic Trees

PhylogenyUz reads Newick [18], Nexus or NHX formatted tree files. To load tree files, select Load Tree menu item from context menu File. PhylogenyUz reconstructs phylogenetic tree immediately without any calculation and shows on main view.

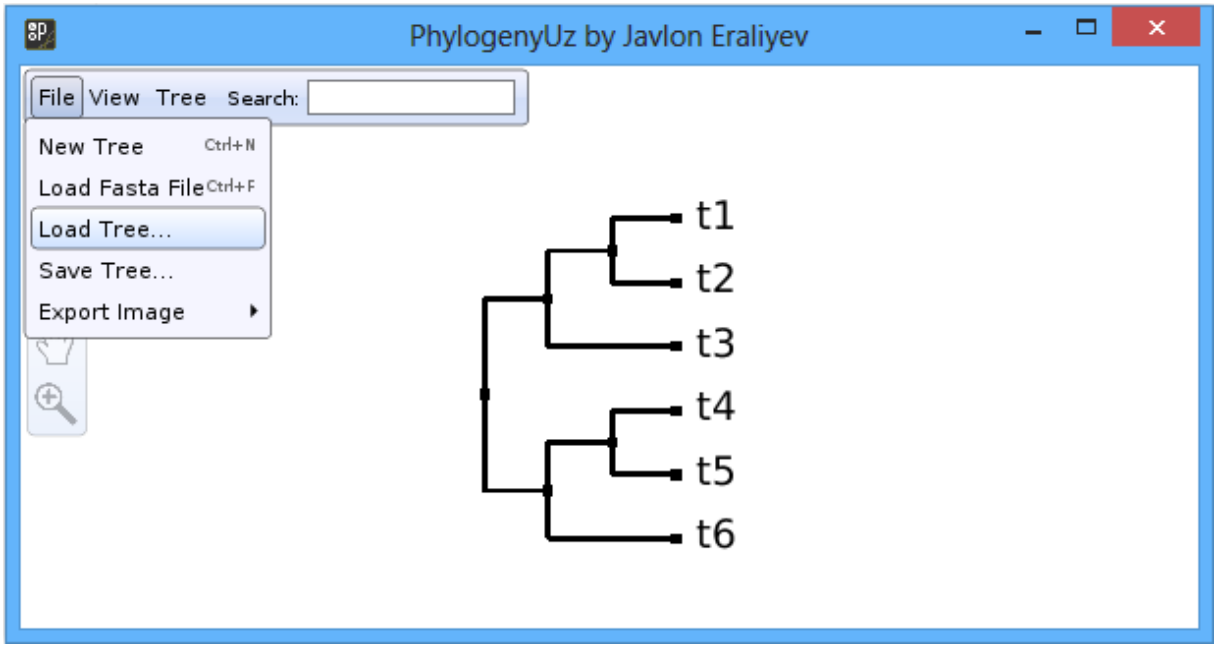

Figure 3.4. Loading phylogenetic tree in PhylogenUz

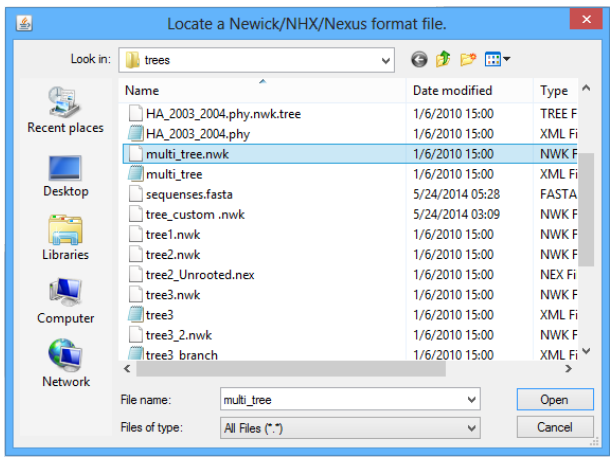

Figure 3.5. Open file Dialog

Saving Phylogenetic Trees

PhylogenyUz saves Newick [18] formatted tree files. To save tree files, select Save Tree menu item from context menu File. It saves tree from main view.

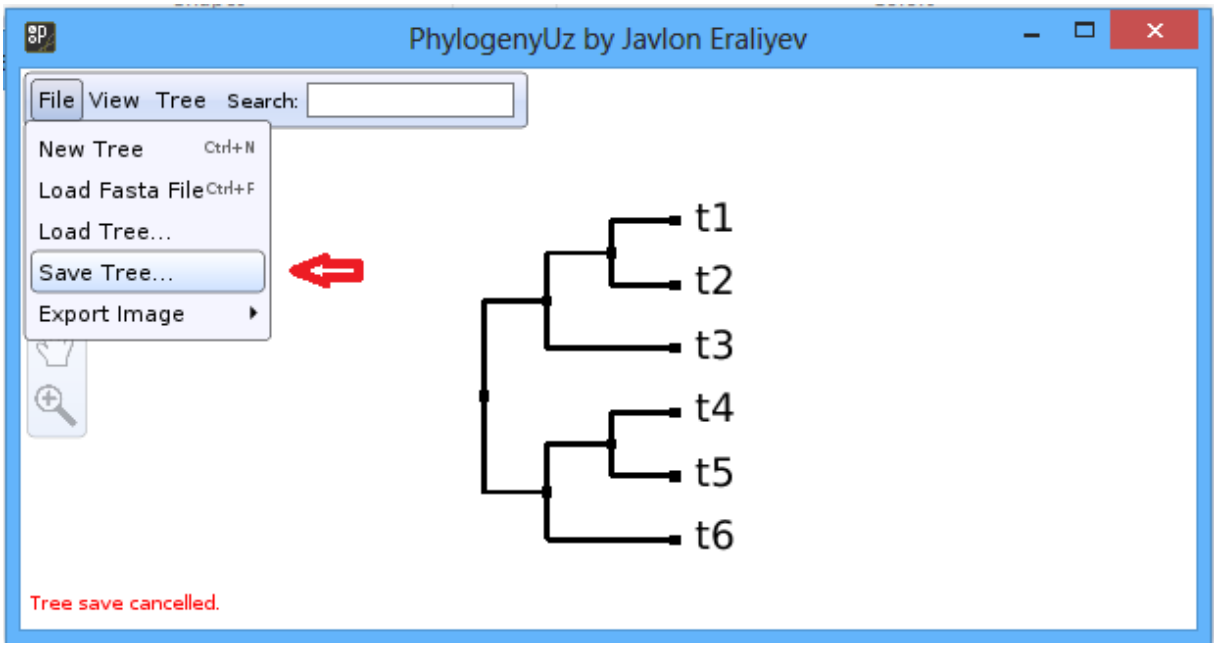

Figure 3.6. Saving phylogenetic tree in PhylogenUz

Saving as Image or PDF

By using this sub menus you can save phylogenetic tree as a image file or PDF file.

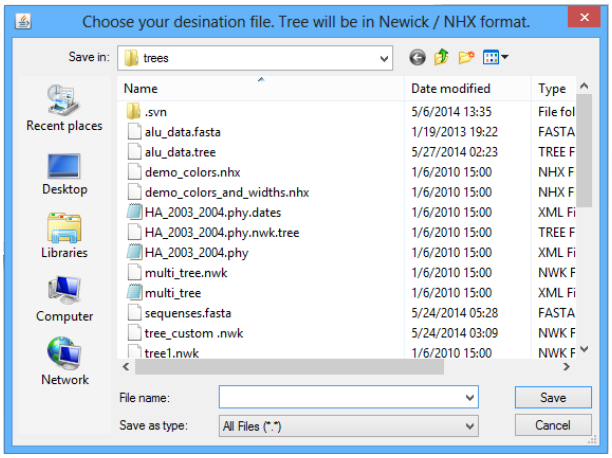

Figure 3.7. Save Dialog

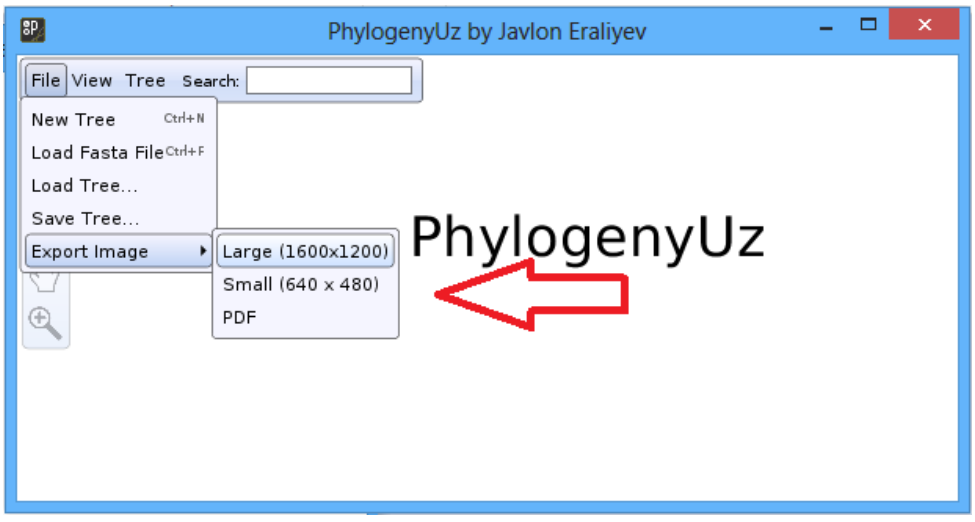

Figure 3.8. Saving as Image or PDF

Render type

By default, Phylogeny displays phylogenetic trees in rectangular form. But user can change render type by selecting menu item Renderer Type from context menu View. It supports 3 types of rendering: Rectangular, Diagonal and Circular. Shurtcuts are Ctrl+1, Ctrl+2 and Ctrl+3 respectively.

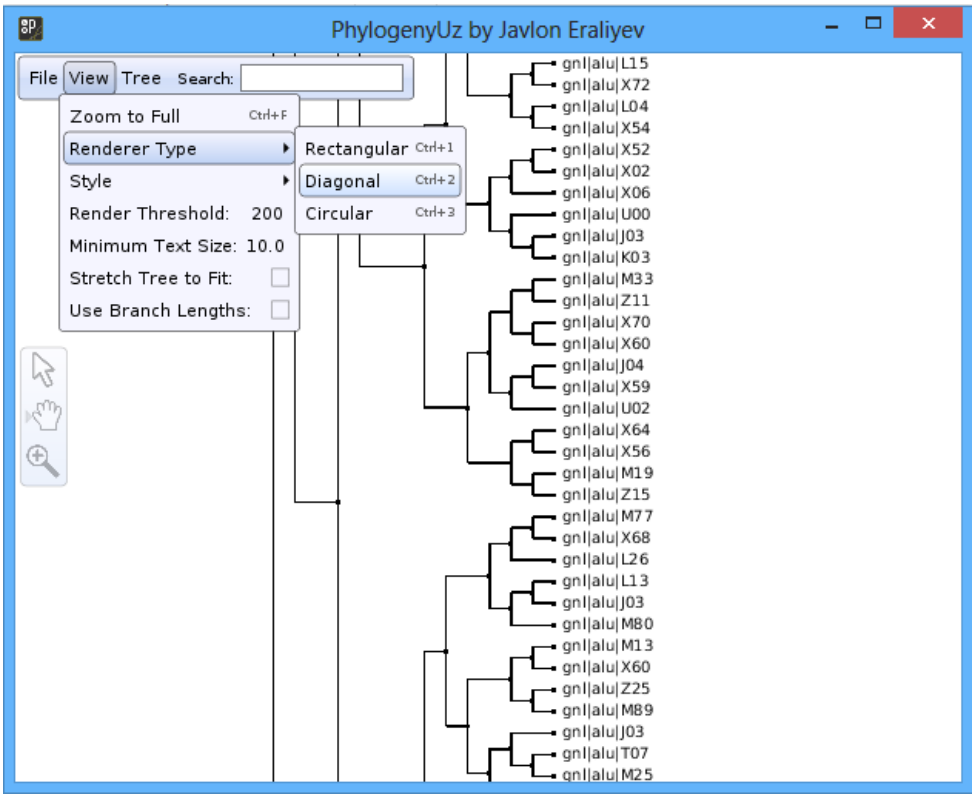

Figure 3.9. Rectangular render type

Styling options

User can change rendering styles as Text Multiplier, Text Angel, Node Size, Line Weight, Render Threshold, Text Size, Stretch Tree and Branch Lengths.

Also, PhylogenyUz can manage tree structure. User can use features as Flip Tree, Enforce Unique Labels, Auto-sort Tree and Remove Elbows.

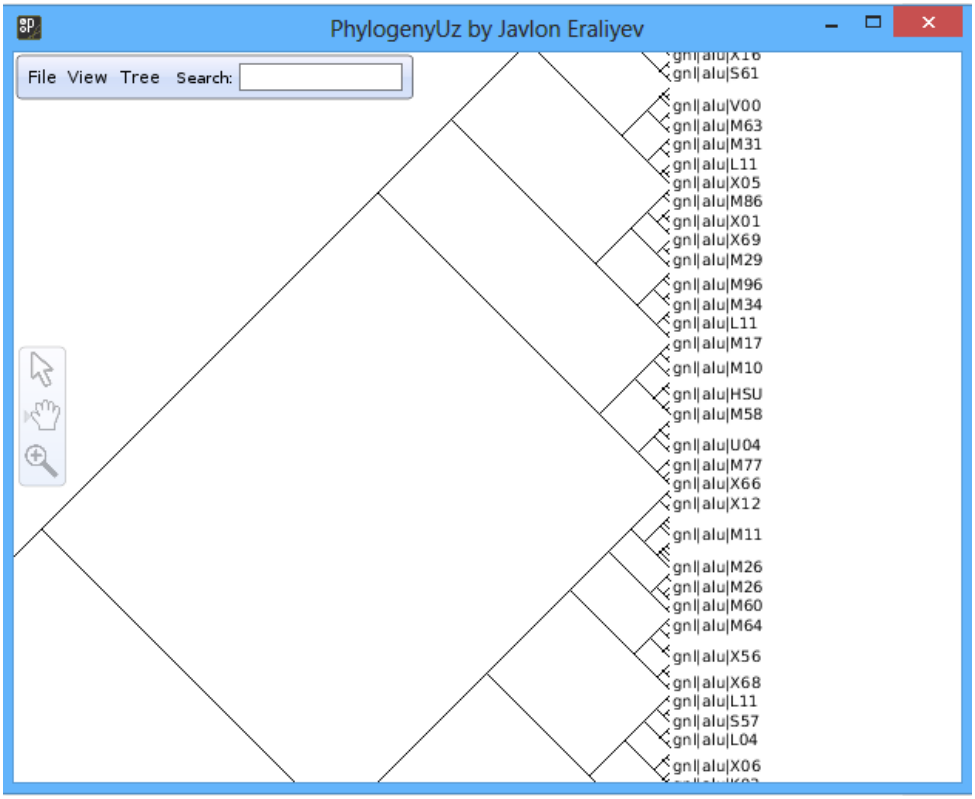

Figure 3.10. Diagonal render type

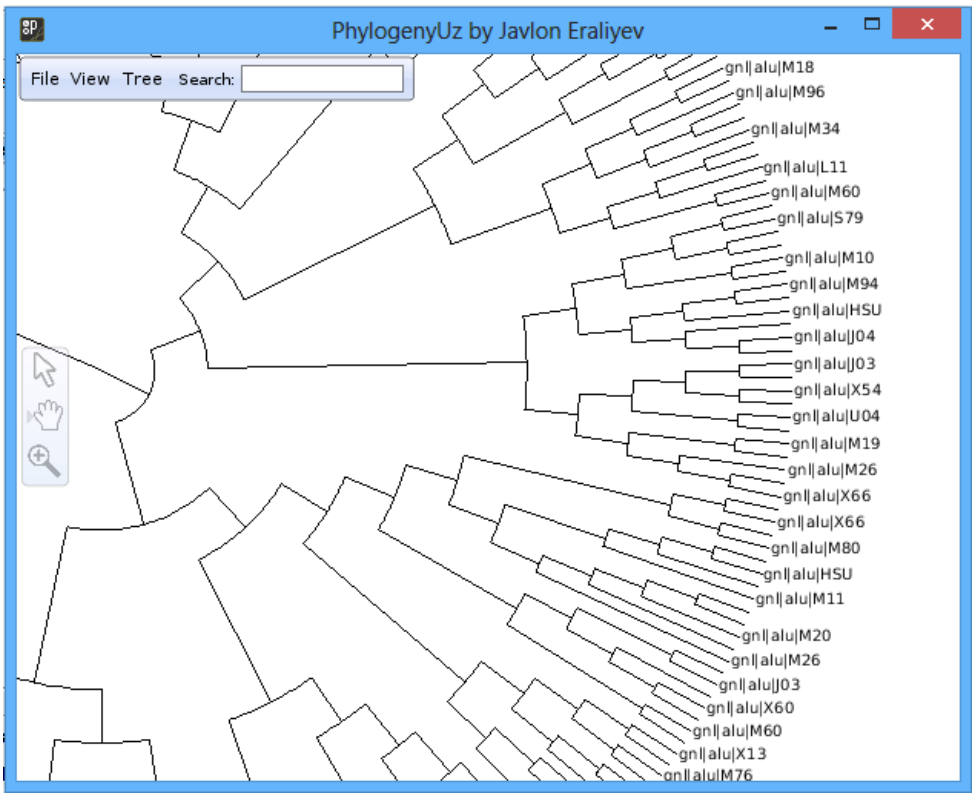

Figure 3.11. Circular render type

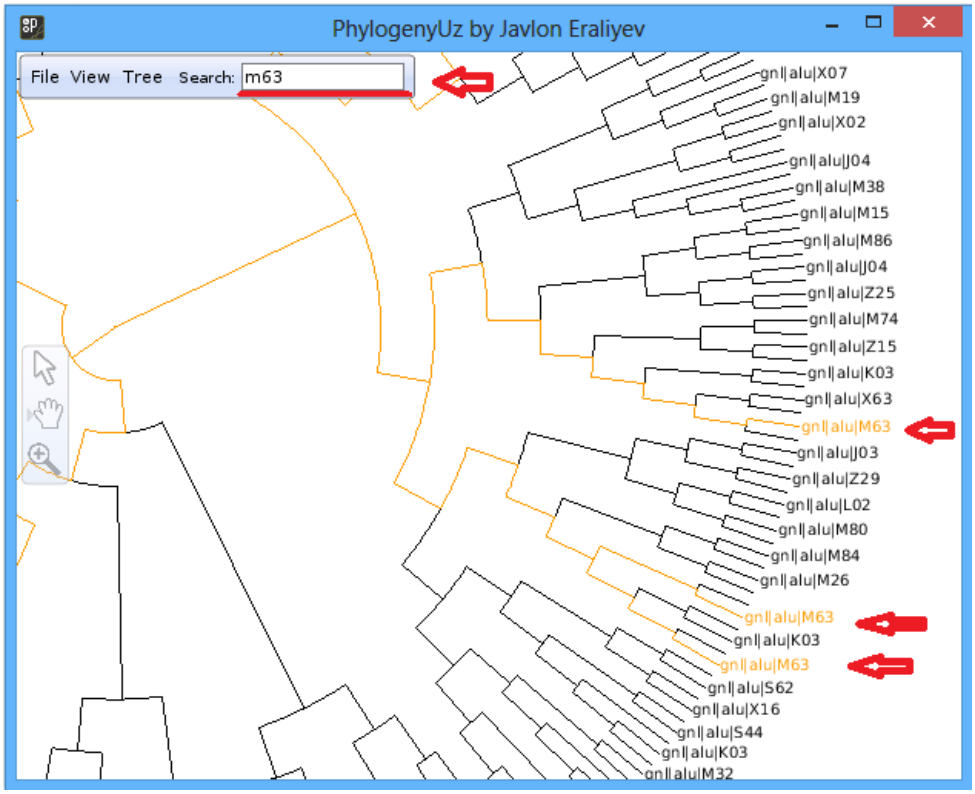

Figure 3.12. Search

Search

Search feature is very useful to find and highlight tree nodes by keyword. Enter keyword into search input box and PhylogenyUz highlights all matched nodes.

3. System Requirements Specification

Introduction. The section introduces the system requirements specification (SRS) for the PhylogenyUz (PU) system to its readers.

Specification Definition. This specification documents the system-level requirements for the PU system.

Specification Objectives. The objectives of this specification of the PU are to:

- Provide a system overview of the PU including definition, goals, objectives, context, and major capabilities.

- Functional requirements.

- Data requirements.

- Quality requirements.

- Constraints.

Intended Audiences. The intended audiences of stakeholders for this specification of the PU include:

- Biology scientists.

- Biology researches.

- Users, who are any private individuals that take part in a biology held in the PU.

External Software. The system requires Java 1.6 or newer version.

Input formats.

Fasta format [17]

A sequence in FASTA format begins with a single-line description, followed by lines of sequence data. The description line is distinguished from the sequence data by a greater-than (">") symbol in the first column. The word following the ">" symbol is the identifier of the sequence, and the rest of the line is the description (both are optional). There should be no space between the ">" and the first letter of the identifier. It is recommended that all lines of text be shorter than 80 characters. The sequence ends if another line starting with a ">" appears; this indicates the start of another sequence. A simple example of one sequence in FASTA format:

>gi|5524211|gb|AAD44166.1| cytochrome b
[Elephas maximus maximus]
LCLYTHIGRNIYYGSYLYSETWNTGIMLLLITMATAFMGYVLP
WGQMSFWGATVITNLFSAIPYIGTNLVEWIWGGFSVDKATLNR
FFAFHFILPFTMVALAGVHLTFLHETGSNNPLGLTSDSDKIPF
HPYYTIKDFLGLLILILLLLLLALLSPDMLGDPDNHMPADPLN
TPLHIKPEWYFLFAYAILRSVPNKLGGVLALFLSIVILGLMPF
LHTSKHRSMMLRPLSQALFWTLTMDLLTLTWIGSQPVEYPYTI
IGQMASILYFSIILAFLPIAGXIENY

Output formats.

Newick format [18]

In mathematics, Newick tree format (or Newick notation or New Hampshire tree format) is a way of representing graph-theoretical trees with edge lengths using parentheses and commas.

(,,(,));	no nodes are named
(A,B,(C,D));	leaf nodes are named
(A,B,(C,D)E)F;	all nodes are named
(:0.1,:0.2,(:0.3,:0.4) :0.5) ;	all but root node have a distance to parent
(:0.1,:0.2,(:0.3,:0.4) :0.5) :0.0;	all have a distance to parent
(A:0.1,B:0.2,(C:0.3,D:0.4):0.5) ;	distances and leaf names (popular)
(A:0.1,B:0.2,(C:0.3,D:0.4)E:0.5)F;	distances and all names
((B:0.2,(C:0.3,D:0.4)E:0.5)F:0.1)A;	a tree rooted on a leaf node (rare)

- Tree: The full input Newick Format for a single tree
- Subtree: an internal node (and its descendants) or a leaf node
- Leaf: a node with no descendants
- Internal: a node and its one or more descendants
- BranchSet: a set of one or more Branches
- Branch: a tree edge and its descendant subtree.
- Name: the name of a node
- Length: the length of a tree edge.

Nexus format. Nexus file format (usually .nex or .nxs) is widely used in Bioinformatics. Several popular phylogenetic programs such as Paup*, MrBayes, Mesquite, and MacClade use this format.

Command inside square brackets [and] are ignored (comment). Each block starts with BEGIN block_name; and finishes with END;

Example 1:

```
#NEXUS
Begin data;
Dimensions ntax=4 nchar=15;
Format datatype=dna symbols="ACTG" missing=? gap=-;
```

```
Matrix
Species1      atgctagctagctcg
Species2      atgcta??tag−tag
Species3      atgttagctag −tgg
Species4      atgttagctag −tag
;
End;
```

Example 2:

```
#NEXUS
#BEGIN TAXA;
  TAXLABELS A B C;
END;

BEGIN TREES;
  TREE tree1 = ((A,B),C);
END;
```

Conclusion for Chapter 3

- Integration algoritms of estimate colculations into reconstructing phylogenetic tree increases quality of results.

- Developed software PhylogenyUz can be used to reconstruct phylogenetic trees.

- Open-source library BioJava has wide scope to analyze phylogenetic data.

Conclusion

This chapter set out to survey large-scale phylogeny and alignment estimation. As we have seen, large-scale alignment and phylogeny estimation (whether of species or of individual genes) is a complicated problem. Despite the multitude of methods for each step, the estimation of very large sequence alignments or gene trees is still quite diffcult, and the estimation of species phylogenies (whether trees or networks!), even more so. Furthermore, the challenges in large-scale estimation are not computational feasibility (running time and memory), as inferential methods can differ in their theoretical and empirical performance.

The aim of this paper is to enable those who have never reconstructed a phylogeny to do so from scratch. The paper does not attempt to be a comprehensive theoretical guide, but describes one rigorous way of obtaining phylogenetic trees. Those who follow the methods outlined should be able to understand the basic ideas behind the steps taken, the meaning of the phylogenetic trees obtained and the scope of questions that can be answered with phylogenetic methods. The protocols have been successfully tested by volunteers with no phylogenetic experience.

Conclusions made in the thesis:

- Algorithms of estimate calculations can be used to analyze phylogenetic data (Figure 1.6).

- Phylogenetic tree reconstruction can be divided to 3 main stages.

- Phylogenetics play an important role in biology

- Main goal of the phylogeny is to reconstruct phylogenetic tree.

- Phylogenetic tree represents evolutionary histories of living organisms.

- Alignment methods vary in type of data (DNA, RNA, or aminoacid) they can handle

- There are three main stages to reconstruct phylogenetic tree: sequences aligment, calculate distance matrix, reconstruction tree.

- Neighbor joining is the simplest methods for reconstructing phylogenetic trees.

- Maximum parsimony sometimes performing better on large trees with high rates of evolution, even though the reverse generally holds for smaller trees.

- Integration algoritms of estimate colculations into reconstructing phylogenetic tree increases quality of results.

- Developed software PhylogenyUz can be used to reconstruct phylogenetic trees.

- Open-source library BioJava has wide scope to analyze phylogenetic data.

References

1. I. A. Karimov. *Mamlakatni modernizatsiya qilish va iqtisodiyotimizni barqaror rivojlantirish yo'lida*. Uzbekiston, 2008. 34-36.
2. I. A. Karimov. *Vatan ravnaqi uchun har birimiz mas'ulmiz*. Uzbekiston, 2001. 98-101.
3. Jesse L. Mecham. Jumpstarting phylogenetic searches. Master's thesis, Brigham Young University, 2006. 1-5.
4. Alkim Ozaygen. Phylogenetic supertree construction using constraint programming. Master's thesis, Cankaya University, 2006. 3-31.
5. Kamilov M.M. Zhuravlev, Yu.I. and Sh.E. Tulyaganov. *Algorithms for Estimate Calculation and Their Application*. Fan, first edition, 1974. 32-41.
6. I. B. Gurevich and A. V. Nefyodov. Algorithms for estimate calculations designed for the case of 2d support sets. part 1: Rectangular support sets. *Cybernetics*, 2001. 662-689.
7. Yu.I. Zhuravlev. *Izbrannie trudi*. Magistr, first edition, 1998. 23-33.
8. Ryan M. Potter. Constructing phylogenetic trees using multiple sequence alignment. Master's thesis, University of Washington, 2008. 2-12.
9. Tandy Warnow. Large-scale multiple sequence alignment and phylogeny estimation. *University of Texas at Austin, Austin, TX, USA*, 2013. 4-22.
10. http://www.clustal.org/clustal2/(Clustalwebsite). [Online; accessed 01-May-2014].
11. Tandy Warnow Usman W. Roshan, Benard M. E. Moret and Tiani L. Williams. A fast algorithmic technique for reconstructing large phylogenetic trees. *Computational Systems Bioinformatics Conference, 2004. CSB 2004*, 2004. 98-109.
12. Hall B.G. Comparison of the accuracies of several phylogenetic methods using protein and dna sequences. *Mol Evol Biol*, 2005. 792-802.

13. David Posada Steven M. Woolley. A comparison of phylogenetic network methods using computer simulation. *Plos One*, 2008. 33-46.
14. Freudenstein J Simmons M. The effects of increasing genetic distance on alignment of, and tree construction from, rdna internal transcribed spacer sequences. *Mol Phylo Evol*, 2003. 444-451.
15. http://biojava.org/wiki/Main_Page(BioJava:anopen-sourceframeworkforbioinformatics). [Online; accessed 5-May-2014].
16. http://www.phylowidget.org/(PhyloWidgetwebsite). [Online; accessed 10-May-2014].
17. https://en.wikipedia.org/wiki/FASTA_format(Fastaformat). [Online; accessed 10-June-2014].
18. https://en.wikipedia.org/wiki/Newick_format(Newickformat). [Online; accessed 09-June-2014].
19. Marco Salemi Philippe Lemey and Anne-Mieke Vandamme. *The Phylogenetic Handbook: a Practical Approach to Phylogenetic Analysis and Hypothesis Testing*. Cambridge University Press, 2009. 22-46.
20. https://en.wikipedia.org/wiki/Phylogenetic_tree(Phylogenetictree). [Online; accessed 11-May-2014].
21. https://en.wikipedia.org/wiki/Distance_matrices_in_phylogeny(Distancematricesinphylogeny). [Online; accessed 11-May-2014].
22. Chuang Peng. Distance based methods in phylogentic tree construction. Master's thesis, Morehouse College, 2011. 3-16.
23. Fabio Pardi. *Algorithms on Phylogenetic Trees*. PhD thesis, St Catharine's College, 2009. 12-20.
24. Xamdamov N.Ya. Xudayberdiyev M.X., Eraliyev J.R. Parametric models partial precedent type algorithms for calculating estimates and their properties. *XIV International Conference "Computer Science: Problems, method-*

ology, technology" and V International School-Conference "Informatics in Education", 2014.

25. Tursunaliyev N. Abdusattarov U., Eraliyev J. Face recognizing from video stream by using viola-jones algorithm. *Innovation ideas, technologies and practical implementation of projects*, 2014.

Appendices

Loding fasta format and saving distanse matrix.

PhyloUI.java

```
//@Author Gregory Jordan

package org.phylowidget.ui;

import com.joewandy.bioinfoapp.model.core.Sequence;
import com.joewandy.bioinfoapp.model.core.io.FastaReader;
import org.andrewberman.ui.EventManager;
import org.andrewberman.ui.FocusManager;
import org.andrewberman.ui.ShortcutManager;
import org.andrewberman.ui.UIGlobals;
import org.andrewberman.ui.menu.MenuItem;
import org.andrewberman.ui.menu.ToolDock;
import org.andrewberman.ui.menu.Toolbar;
import org.biojava3.alignment.Alignments;
import org.biojava3.alignment.template.Profile;
import org.biojava3.core.sequence.AccessionID;
import org.biojava3.core.sequence.DNASequence;
import org.biojava3.core.sequence.MultipleSequenceAlignment;
import org.biojava3.core.sequence.compound.NucleotideCompound;
import org.biojava3.phylo.ProgessListenerStub;
import org.biojava3.phylo.TreeConstructionAlgorithm;
import org.biojava3.phylo.TreeConstructor;
import org.biojava3.phylo.TreeType;
import org.forester.evoinference.matrix.distance.DistanceMatrix;
import org.phylowidget.PhyloTree;
import org.phylowidget.PhyloWidget;
import org.phylowidget.net.NodeInfoUpdater;
import org.phylowidget.net.SecurityChecker;
import org.phylowidget.render.BasicTreeRenderer;
import org.phylowidget.render.NodeRange;
import org.phylowidget.tree.CachedRootedTree;
import org.phylowidget.tree.PhyloNode;
import org.phylowidget.tree.RootedTree;
import org.phylowidget.tree.TreeIO;
import processing.core.PApplet;
```

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.*;
import java.lang. reflect . Field ;
import java.net.URL;
import java.text.DecimalFormat;
import java.util . ArrayList ;
import java.util .HashMap;
import java.util . List ;

import org.andrewberman.ui.FocusManager;

public class PhyloUI implements Runnable {

   // ...

   // Loads fasta format file and builds tree
   public void fastaLoad() {
      FileDialog fd = new FileDialog(PhyloWidget.ui.getFrame(),
            "Locate␣a␣Fasta␣format␣file.", FileDialog.LOAD);
      fd.pack();
      fd. setVisible (true);
      String directory = fd. getDirectory ();
      String filename = fd. getFile ();
      if (filename == null) {
         PhyloWidget.setMessage("Fasta␣load␣cancelled.");
         return;
      }
      final File f = new File( directory , filename );
      setMessage("Loading␣fasta...");
      new Thread() {
         public void run() {

            FastaReader fastaReader = new FastaReader(
                  FastaReader. FastaFileContent .DNA_SEQUENCE, f.getPath());

            List<Sequence> list = fastaReader . processFile ();
```

```
List<DNASequence> sequences = new ArrayList<DNASequence>();

for (Sequence sequence :  list ) {
    DNASequence dnaSequence = new DNASequence(
            sequence.getSequenceString());
    dnaSequence.setAccession(new AccessionID(sequence.getId()));

    sequences.add(dnaSequence);
}

 Profile <DNASequence, NucleotideCompound> profile = Alignments
        .getMultipleSequenceAlignment(sequences);

MultipleSequenceAlignment<DNASequence, NucleotideCompound>
      multipleSequenceAlignment = new MultipleSequenceAlignment<DNASequence,
      NucleotideCompound>();

org.biojava3.core.sequence.template.Sequence<NucleotideCompound> sequence;
DNASequence nucleotideCompounds;
for (int i = 1; i < profile.getSize() + 1; i++) {
    sequence = profile.getAlignedSequence(i);
    nucleotideCompounds = new DNASequence(
            sequence.getSequenceAsString(),
            sequence.getCompoundSet());
    nucleotideCompounds.setAccession(sequence.getAccession());
    multipleSequenceAlignment
            .addAlignedSequence(nucleotideCompounds);
}

TreeConstructor<DNASequence, NucleotideCompound> treeConstructor = new
      TreeConstructor<DNASequence, NucleotideCompound>(
        multipleSequenceAlignment,  TreeType.NJ,
        TreeConstructionAlgorithm.PID,
        new ProgessListenerStub());

String  newick = null;
try {
    treeConstructor .process();
    newick =  treeConstructor .getNewickString(true, true);
```

```
            } catch (Exception e) {
                e. printStackTrace () ;
            }

            if (newick == null) {
                return;
            }

            PhyloTree t = (PhyloTree) TreeIO.parseNewickString(
                    new PhyloTree(), newick);
            p.noLoop();
            if (t != null) {
                PhyloWidget. trees . setTree (t,
                        treeConstructor . getDistanceMatrix () ) ;
                setMessage("");
            } else {
                setMessage("Error loading tree!");
            }
            p.loop () ;
            layout () ;
        }
    }. start () ;
}

// Shows distance matrix
public void showMatrix() {

    DistanceMatrix matrix = PhyloWidget. trees . getDistanceMatrix () ;

    if (matrix == null) {
        JOptionPane.showMessageDialog(this.p, "Avvalo daraxt yasash kerak");
        return;
    }

    JFrame window = new JFrame("Distance Matrix",
            p. getGraphicsConfiguration () ) ;

    window.setLayout(new GridLayout(matrix.getSize () , matrix . getSize () ) ) ;
```

```
        for (int i = 0; i < matrix.getSize(); i++) {
            for (int j = 0; j < matrix.getSize(); j++) {
                window.add(new JLabel(""
                        + new DecimalFormat("#.###").format(matrix.getValue(i,
                                j))));
            }
        }

        window.pack();
        window.setVisible(true);

    }

    // ...
}
```

Configuring language.

PhyloConfig.java

```
package org.phylowidget.ui;

import org.andrewberman.ui.Color;
import org.andrewberman.ui.unsorted.MethodAndFieldSetter;
import org.phylowidget.PhyloWidget;
import org.phylowidget.tree.RootedTree;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.URL;
import java.util.HashMap;

public class PhyloConfig {

    // ...

    /*
     * Choose the preset XML menu files which PhyloWidget will load.
     *
     * You can load up multiple XML menu files by giving a semicolon-
        delimited list. See the default
```

```
     * value for an example of this.
     *
     * You may also simply let this string be the XML data which you
        want to load. This is useful for
     * demonstration purposes, letting the user edit and change the
        menu structure in real-time.
     *
     * Core menu definitions:
     *  - "context.xml"   The context menu which appears when you click
         a node.
     *  - "dock.xml"      The dock, which holds the arrow, pan, and
        zoom tools.
     *  - "toolbar.xml"   The toolbar, which sits at the top of the
        screen and contains
     *                  the menu items and search bar.
     *  - "none.xml"      No menus at all.
     *
     * Some useful variants:
     *  - "toolbar-onlysearch.xml" A toolbar with only the search
        command, nothing else.
     *  - "dock-onlynav.xml"  A dock with only the pan and zoom
        settings, no arrow.
     *  - "context-linkout.xml"  A context menu which lets you link out
         to other sites.
     *                   See the XML file for more information.
     *  - "dock-hidden.xml" A hidden dock. Useful for providing the
        dock's functionality without
     *                   cluttering up the screen. Users can still use
         the keyboard shortcuts
     *                   to switch between tools.
     *  - "toolbar-hidden.xml" Same idea as above, but with the
        toolbar.
     */
    public String menus = "full.xml;";

     //public String menus = "uz.xml;";

     // ...

}
```

UI setting and string file.

full.xml

```
<menuset name="PhyloUz␣Menus">
    <menu type="Toolbar" fontSize="11" x="2" y="2">
        <item name="File">
            <item name="New␣Tree" action="treeNew" shortcut="control-n"/>
            <item name="Load␣Fasta␣File" action="fastaLoad" shortcut="control-f"/
                >
            <item name="Load␣Tree..." action="treeLoad"/>
            <item name="Save␣Tree..." action="treeSave"/>
            <item name="Export␣Image">
                <item name="Large␣(1600x1200)" action="fileOutputBig"/>
                <item name="Small␣(640␣x␣480)" action="fileOutputSmall"/>
                <item name="PDF" action="fileOutputPDF"/>
            </item>
        </item>
        <item name="View">
            <item name="Zoom␣to␣Full" action="viewZoomToFull" shortcut="control-
                F"/>
            <item name="Renderer␣Type">
                <item name="Rectangular" action="viewRectangular" shortcut="
                    control-1"/>
                <item name="Diagonal" action="viewDiagonal" shortcut="control-2"/
                    >
                <item name="Circular" action="viewCircular" shortcut="control-3"/
                    >
            </item>
            <item name="Style">
                <item name="Text␣Multiplier" type="NumberScroller" property="
                    textSize" increment=".02" min="0.05"
                     max="2"/>
                <item name="Text␣Angle" type="NumberScroller" property="
                    textRotation" increment=".5" min="-45"
                     max="45"/>
                <item name="Node␣Size" type="NumberScroller" property="nodeSize"
                    increment="0.05" min="0" max="10"/>
                <item name="Line␣Weight" type="NumberScroller" property="
                    lineSize" increment="0.05" min="0" max="10"/>
            </item>
```

```
            <item name="Render Threshold" type="NumberScroller" property="
                renderThreshold" increment="5" min="5"
                 max="1000"/>
            <item name="Minimum Text Size" type="NumberScroller" property="
                minTextSize" increment=".1" min="0"
                 max="36"/>
            <item name="Stretch Tree to Fit" type="CheckBox" methodCall="
                setStretchToFit"/>
            <item name="Use Branch Lengths" type="CheckBox" methodCall="
                setUseBranchLengths" shortcut="ctrl-b"/>
        </item>
        <item name="Tree">
            <item name="Mutator (for fun!)">
                <item name="Mutate Once" action="treeMutateOnce" shortcut="
                    control-m"/>
                <item name="Mutate Slow" action="treeMutateSlow"/>
                <item name="Mutate Fast" action="treeMutateFast"/>
                <item name="Stop Mutating" action="treeStopMutating" shortcut="
                    control-shift-m"/>
            </item>
            <item name="Enforce Unique Labels" type="CheckBox" methodCall="
                setEnforceUniqueLabels"/>
            <item name="Flip Tree" action="treeFlip" shortcut="control-R"/>
            <item name="Auto-sort Tree" action="treeAutoSort" shortcut="control-
                L"/>
            <item name="Remove Elbows" action="treeRemoveElbows" shortcut="
                control-E"/>
        </item>
        <item type="Spacer" spaceWidth="5"/>
        <item type="org.phylowidget.ui.SearchBox" name="Search:" width="140"/
            >
    </menu>

    <!-- If you want the user to be able to interact with the tree AT
        ALL (including through the context menu),
           then you must have a ToolDock with the associated tools. You
                can set the "hidden" attribute of the
           ToolDock element to "True" if you want to have the tools but
                keep the dock hidden. -->
    <menu name="dock" type="ToolDock" width="30" rotation="left" hidden="false">
```

```
        <!-- the "rotation" parameter can have the values: left, right,
             top, bottom -->
        <item name="Arrow" tool="Arrow" shortcut="a" icon="dock/arrow.png"/>
        <item name="Scroll" tool="Scroll" shortcut="s" icon="dock/grab.png"/>
        <item name="Zoom" tool="Zoom" shortcut="z" icon="dock/zoom.png"/>

        <!-- Look at this nifty methodcall tag! Not worth the effort,
             you say? I probably agree. -->
        <methodcall method="selectTool" param="Arrow"/>
    </menu>

    <menu name="context" type="org.phylowidget.ui.PhyloContextMenu">
        <item name="Tree Edit" hint="t">
            <item name="Add" hint="a">
                <item name="Child" action="nodeAddChild" hint="c"/>
                <item name="Sister" action="nodeAddSister" hint="s"/>
            </item>
            <item name="Delete" hint="d">
                <item name="This node" action="nodeDelete" hint="t"/>
                <item name="Subtree" action="nodeDeleteSubtree" hint="s"/>
            </item>
        </item>
        <item name="Layout" hint="a">
            <item name="Reroot" action="nodeReroot" hint="r"/>
            <item name="Flip subtree" action="nodeFlipSubtree" hint="f"/>
            <item name="Switch children" action="nodeSwitchChildren" hint="s"/>
        </item>
        <item name="Clipboard" hint="c">
            <item name="Cut" action="nodeCut" hint="x"/>
            <item name="Copy" action="nodeCopy" hint="c"/>
            <item name="Paste" action="nodePaste" hint="v"/>
            <item name="Swap" action="nodeSwap" hint="s"/>
            <item name="Clear" action="nodeClearClipboard" hint="r"/>
        </item>
        <item name="Node Edit" hint="e">
            <item name="Name" action="nodeEditName" hint="n"/>
            <item name="Branch length" action="nodeEditBranchLength" hint="b"/>
        </item>
    </menu>
</menuset>
```