UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR
THEORETISCHE INFORMATIK

Implementation and Comparison of Algorithms for Constructing and Visualizing Phylogenetic Trees

Implementierung und Vergleich von Algorithmen
für die Konstruktion und Visualisierung
phylogenetischer Bäume

Bachelorarbeit

im Rahmen des Studiengangs
Molecular Life Science
der Universität zu Lübeck

vorgelegt von
Sarah Maya Mäusle

ausgegeben und betreut von
Prof. Dr. Till Tantau

mit Unterstützung von
Dipl.-Inf. Christoph Stockhusen,
Oliver Witt, M.Sc.

Lübeck, den 12. Dezember 2012

reviewer/gutachter

Prof. Dr. Till Tantau
Institut für Theoretische Informatik
Universität zu Lübeck

and/und

PD Dr. Hanns-Martin Teichert
Institut für Mathematik
Universität zu Lübeck

DECLARATION

I hereby declare that this thesis has been composed by myself and that I have taken use of no other resources than the ones stated.

ERKLÄRUNG

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe.

Lübeck, den 12. Dezember 2012 ________________________

ABSTRACT

TEX is a typesetting system which is frequently used for scientific documents. The TEX package *Ti*k*Z* allows for the creation of high quality vector graphics directly from a TEX document. The aim of this thesis was to expand the Ti*k*Z graph drawing library, in order to enable the computation and visualization of *phylogenetic trees*.

Phylogenetic trees are diagrams that display the *phylogeny* of a set of taxa in a tree-like manner. One type of approach on determining the evolutionary history of a dataset are the *distance-based* methods, where the phylogeny is estimated on the basis of a distance matrix. There are a number of different distance-based methods, of which two are dealt with here: the UPGMA and the BME algorithm. While the former has the advantage of being simple and thus easy to implement, the latter is superior when it comes to topological accuracy, time complexity, and applicability. As both these algorithms do not only determine the tree's topology, but also the respective edge lengths, a graph drawing algorithm, which can handle fixed edge lengths, is also necessary. Both the phylogenetic tree and the graph drawing algorithms have been implemented in the programming language *Lua*, as LuaTEX delivers the possibility of incorporating Lua-code directly into TEX. The algorithms and their implementation will be discussed and compared, whereas the implemented module is used for the exemplary computation and visualization of phylogenetic trees directly in this document.

ZUSAMMENFASSUNG

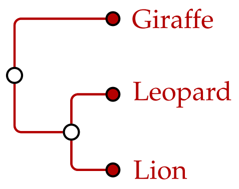
TEX ist ein Textsatzsystem, das häufig für die Erstellung wissenschaftlicher Dokumente eingesetzt wird. Das TEX-Paket *TikZ* ermöglicht zusätzlich die Erzeugung von qualitativ hochwertigen Vektorgraphiken direkt in einem TEX-Dokument. Das Ziel dieser Arbeit war es, die Ti*k*Z-Graph-Drawing-Bibliothek so zu erweitern, dass die Berechnung und Visualisierung *phylogenetischer Bäume* ebenfalls möglich wird.

Phylogenetische Bäume sind Diagramme, die die *Phylogenie* einer Menge an Taxa in einer baumartigen Struktur darstellt. Die Stammesgeschichte kann z.B. durch die sogenannten *distance-based methods* (engl. distanzbasierte Methoden) berechnet werden, wobei die Phylogenie anhand einer Distanzmatrix abgeschätzt wird. Es gibt verschiedene distanzbasierte Ansätze, von denen zwei in dieser Arbeit näher betrachtet werden: der UPGMA- und der BME-Algorithmus. Während UPGMA eher einfach und dadurch auch leicht zu implementieren ist, ist BME deutlich überlegen, wenn es um akkurate Topologie, Zeitkomplexität und Einsetzbarkeit geht. Da die beiden Algorithmen nicht nur die Topologie des Baumes, sondern auch dessen Kantenlängen bestimmen, ist auch ein Algorithmus erforderlich, der mit dem Zeichnen von Graphen mit festen Kantenlängen umgehen kann. Sowohl die Algorithmen für die Berechnung phylogenetischer Bäume als auch die für deren Zeichnung wurden in der Programmiersprache *Lua* implementiert, da es LuaTEX ermöglicht, Lua-Code direkt in TEX einzubetten. Die Algorithmen und ihre Implementierungen werden diskutiert und verglichen, wobei das implementierte Modul für die Berechnung und Visualisierung einiger beispielhafter phylogenetischer Bäume direkt in diesem Dokument angewandt wird.

TABLE OF CONTENTS

1 INTRODUCTION

TEX is a typesetting program, which, together with its extensions such as *LATEX* and *BibTEX*, has several advantages over standard word processors, as for example the facilitated correct display of mathematical formulas, the automatic generation of bibliographies, and the possibility of easily referencing to figures, tables, and literature sources. Thus, TEX is often used for writing scientific papers or textbooks. Scientific documents dealing with phylogenetical topics often require the calculation and illustration of *phylogenetic trees*. If the author of such a document so far wished to display phylogenetic trees in TEX, he or she had to resort to include them as pictures, generated by other programs. This, of course, is possible, but brings along the disadvantage of not being able to dynamically modify the trees in retrospective, both content- and appearance-wise, without reverting to the other programs used for computing and visualizing them in the first place. Alternatively, the author can also have the tree computed by another program and then draw it directly in the TEX document, for example with Ti*k*Z, but then again a change in content would be dependent on the other program, as phylogenetic trees are often computed by complex algorithms. The general goal of this thesis was to change this inconvenience by delivering the possibility of creating phylogenetic trees directly in a TEX document on the basis of evolutionary input data: Typing a few lines of TEX code should then result in something like the tree on the right, which has indeed been produced in this document by doing so. Before looking at how this is possible, it should first be clarified what the *significance* of a pyhlogenetic tree is, both from a *biological* and a *bioinformatical* point of view.



1.1 *What are Phylogenetic Trees?*

Evolutionary studies determining the historical development of the relations between organisms or other biological systematic units, form the field of *phylogenetics*. It has long since been of scientific interest to determine how closely related organisms are and when they diverged from a common ancestor. Before information about molecular structures, especially deoxyribonucleic acid (DNA), became available as a basis of comparison, morphologic differences where used to classify organisms. It was difficult, if not impossible, to always distinguish between analogous and homologous structures, and thus determine the organisms' true *phylogeny*, i.e. its evolutionary history. Nowadays, biochemical methods, such as methods for DNA sequencing, are standard procedures for gaining insight into molecular genetics, offering new possibilities for phylogenetical studies: By comparing the sequences of homologous genes, the relation between organisms may be estimated. While some genes have been shown to mutate at a rather quick rate, other genes retain practically the same sequence over millions of generations. This renders the possibility of analyzing both recent evolutionary events and such that took place a long time ago [3].

A *phylogenetic tree* is a diagram displaying the phylogeny between taxa in a tree-like manner, see Figure 1.1 for an example. A *taxon* is a systematic unit, which may be a single gene, an organism or a group of organisms with a common ancestor [2]. Thus, to name an example, an individual animal with its individual set of chromosomes can be a taxon, but also the lion species (*Panthera leo*), and the family of cats (Felidae), may be chosen as taxa. In phylogenetic trees these taxa are represented by leaves. Every inner node, on the other hand, embodies an evolutionary event giving rise to two or more distinct lineages, although it is important to note that such a node does not necessarily suggest a real common ancestor at that point in evolution. The edge lengths may or may not be of significance; if they do matter, they may either depict the time that has passed between the two evolutionary events or the difference between them, i.e. the rate of mutation.

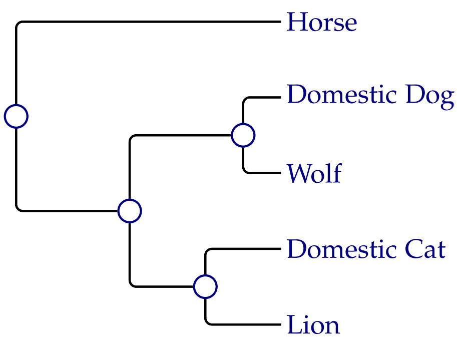


Figure 1.1: An exemplary phylogenetic tree. The edge lengths do not correspond to the true evolutionary distances of the displayed taxa.

The fact that sequencing genes—or even whole genomes—has recently become increasingly cheaper and faster, has made it possible to compare more and more taxa on a molecular level. This constantly rising amount of data makes a manual interpretation impractical if not impossible, and thus computer programs have been developed for this problem. Various different algorithms have been designed to compute phylogenetic relationships and to display them as trees, bringing us to the bioinformatical aspect of this topic.

There are many different approaches to computing phylogenetic trees. They differ, on the one hand, in the type of data used as the foundation for estimating the evolutionary distance and, on the other hand, in how the data is processed. One can roughly distinguish between two main data sources: the comparison of whole genomes or parts of them, and the comparison of individual *characters*, as for example described in [2].

The main task of phylogenetic tree algorithms is the computation of the tree's *topology*, i.e. its inner structure, and, if required, the computation of the edge lengths. In general there are two main groups of computational approaches on calculating phylogenetic trees: the *sequence-based methods* and the *distance-based methods* [7]. The former are based on—and restricted to—the sequence alignment of genes or (parts of) the genomes of different taxa. The results of these alignments are then analyzed for an evolutionary model which may explain them. For this there are again different approaches. One of them is the *maximum parsimony method* [7], which operates by searching for the tree that explains the evolutionary events in the easiest way. This is done by searching for hypothetical sequences of the hypothetical ancestors—denoted by the inner nodes—that assume the least mutations of their progeny. The *distance-based* approach, on the other hand, calculates the phylogenetic tree on the basis of a *distance matrix*, which contains the information of how closely related all the taxa are to each other. This distance matrix can be obtained by sequence alignment methods, but also by comparing any other given data. Both approaches have their advantages and disadvantages: while the former provides methods focused on sequence alignment and is thus superior in that area, the latter is less specialized, but also more widely applicable and in general much faster concerning the run time [7]. Due to this wide applicability, the focus here lies on the distance-based methods.

Within the limits of this thesis two such approaches have been implemented: The first and rather simple approach is the *Unweighted Pair Group Method with Arithmetic Mean* (UPGMA), first described by Sokal and Michener in 1958 [12]. Here the tree is constructed by iteratively forming clusters and merging these to form a rooted tree. UPGMA is the progenitor of the slightly more complex neighbor-joining method, which, unlike UPGMA, does not require the distance matrix to be ultrametric—a fact that will be explained in the subsequent section.

The second approach of choice is an even more powerful one than neighbor-joining: Studies by Richard Desper and Olivier Gascuel [4], first published in 2002, have shown that algorithms based on the *minimum-evolution* principle render quite accurate phylogenetic trees. Furthermore, a *balanced minimum evolution* (BME) approach seems to be even more accurate. This algorithm is combined with a *balanced nearest neighbor interchanging* (BNNI) algorithm, in order to further enhance the results, as also shown by Desper and Gascuel.

In addition, it is to be said that the sequence alignment of nucleic acids, which is often needed for calculating the distance matrix, is not trivial and is in itself a sub-area of bioinformatical studies. Algorithms used for such alignments are thus neither part of this thesis, nor of the implemented module; the user's input is restricted to precalculated distance matrices.

1.2 *Phylogenetic Trees in TeX*

Having clarified what phylogenetic trees are, one can go back to the question of how it is possible to enable their computation and visualization in TeX.

First of all, it should be clarified what TeX actually is. TeX is a typesetting system, which was developed by Donald E. Knuth mainly between 1977 and 1989 with the goal of enabling the "creation of beautiful books—and especially for books that contain a lot of mathematics" [9]. Another important aim was to enable the reproduction of a document with exactly the same results on preferably any platform. To a great extend, TeX separates the content of a document from its layout. Every layout option, such as the margin size or setting a word in italics, is specified by a command in the system's own language, also called plain TeX; such a document is thus in principle programmed, a fact that makes the system very powerful. Users are also enabled to program macros freely, adding new options and changing the existing ones. In general, TeX thus offers the possibility of running self-written algorithmic code, an important prerequisite for this thesis. An example for this is *LaTeX*, created by Leslie Lamport, which is basically a large collection of macros that strongly facilitates the usage of TeX [6].

Now, how can the connection between phylogenetic trees and TeX be established? Both the computation and the visualizing of phylogenetic trees—of which the latter is a graph drawing problem—are algorithmic problems, and can be implemented in a programming language. Programming in plain TeX, though, is difficult and not well-suited for such complex algorithms as computing and drawing phylogenetic trees. *Lua*, on the other hand, is a scripting language with an easy-to-read and easy-to-write syntax. It is often integrated into programs written in other programming languages, as the fact that Lua is simple, efficient and portable onto many platforms, is often of great value [8]. LuaTeX and LuaLaTeX [10], first made public in 2007, allow the easy direct incorporation of Lua code into a TeX document.

The TeX package Ti*k*Z, first released by Till Tantau in 2005 [13], is a powerful tool that allows the construction of vector graphics by *programming* them; for example the following line of code

```
\tikz\node[star,draw=red,star point height=.2cm,scale=.5] {};
```

results in a little red star: ☆ . The package contains many already implemented functions that have facilitated the implementation of the phylogenetic tree algorithms, for example the generation of nodes and edges, the visualization of the final graphic, and the syntax for programming such a graphic.

The framework for graph drawing in Ti*k*Z, which facilitates the implementation of graph drawing algorithms in Lua, has first been published by Till Tantau in 2012 [14]. In general, the connections between Ti*k*Z and Lua can be summarized as follows: The author inputs the required data using Ti*k*Z, which creates the nodes and other graph elements. The complete graph information is then passed to the Lua layer, which is where the graph drawing algorithms are invoked. Once Lua has finished the graph computation, the results are handed back to Ti*k*Z, and the graph is adjusted to fit those results. Finally, the graph is displayed in the TEX document. Thus, the usage of Ti*k*Z, Lua, and LuaTEX enabled the creation of the module that allows the computation and visualization of phylogenetic trees directly in a TEX document.

1.3 General Aim of the Thesis and Motivation

The module, implemented as part of this thesis, aims at enabling the user to compute and display phylogenetic trees in TEX. This can be of great support when an author is in need of constructing a phylogenetic tree for a scientific document. He or she neither has to use an external program for the calculation and/or display of the tree, nor does he or she need to draw the tree by hand, for example with Ti*k*Z or with an additional drawing program. Figure 1.2 contains another exemplary tree, and Figure 1.3 shows the Ti*k*Z-code that created this tree: In lines 3–9 the general options are set, i.e. the phylogenetic tree algorithm is chosen, as well as the layout type, and the nodes' and edges' styles are set. The distance scaling factor in line 10 can be used to scale the phylogenetic distances and thus the edge lengths (see Section 3.3). In lines 12–16 the taxa are declared and named, and finally in lines 19–22 the distance matrix is specified. The fact that the tree is directly created within TEX by a few lines of code allows the user to easily change the tree's appearance and/or content at will. The tree can also be used as part of a larger Ti*k*Z picture.

This thesis discusses the two phylogenetic tree algorithms and their implementation. The algorithms will also be tested and compared, and suggestions concerning further extensions and additions to the module will be made.

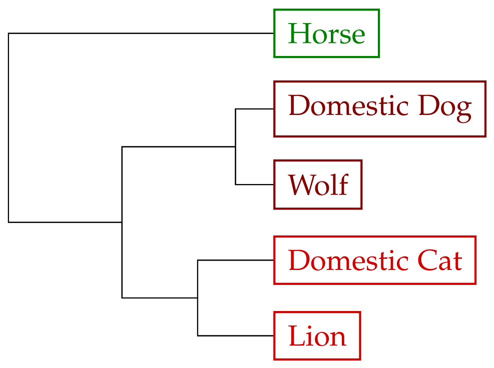


Figure 1.2: The same exemplary phylogentic tree as in figure 1.1, but with different decorative options. The colors additionally underline the closeness of the taxa's relations: all taxa belong to the class Mammalia; the red taxa are of the order Carnivora, whereas the dark red taxa are of the genus Canis, and the light red taxa belong to the family of the Felidae.

```latex
\begin{tikzpicture}
   \graph[
      phylogenetic tree,
      phylogenetic algorithm = UPGMA,
      phylogenetic layout = rectangular phylogram,
      grow = right,
      every phylogenetic inner node/.style = {},
      nodes = {inner sep=5pt, thick, draw, rectangle},
      edges = {draw, rounded corners=2.5pt, thick},
      distance scaling factor = 1cm,
   ]{
         L/Lion [red!80!black],
         C/Domestic Cat [red!80!black],
         W/Wolf [red!50!black],
         D/Domestic Dog [red!50!black],
         H/Horse [green!50!black],

         % distance matrix
         L [distances = {C = 2, W = 4, D = 4, H = 7}];
         C [distances = {       W = 4, D = 4, H = 7}];
         W [distances = {              D = 1, H = 7}];
         D [distances = {                     H = 7}];
   };
\end{tikzpicture}
```

Figure 1.3: The Ti*k*Z-code within this LuaLaTeX document that created the phylogenetic tree displayed in Figure 1.2.

2 COMPUTING PHYLOGENETIC TREES

Many phylogenetic algorithms—as the ones dealt with in this thesis—yield two results: a tree topology, i.e. the arrangement of the nodes with respect to each other and the presence or absence of edges, and the edge lengths.

The topology is the tree's most important characteristic and its computation is the central part of phylogenetic algorithms. Some algorithms, like UPGMA, build up the tree step by step without modifying the already-built parts, while others, like BME, constantly change the tree's topology until the tree has gained its final structure. In contrast to the other two mentioned algorithms, BNNI does not compute a tree from scratch, but merely improves an already existing tree. Phylogenetic trees are often binary trees, i.e. their inner nodes have exactly two children; nevertheless it is also possible for them to have nodes with a higher number of children, see Section 3 on graph drawing algorithms for more detailed information. Phylogenetic trees may be either rooted or unrooted, which depends on the algorithms that compute them.

Often, though not always, phylogenetic trees have fixed edge lengths. If they do, the edge lengths give insight into either *when in time* evolution has taken place or how significant the resulting difference, or rather *distance*, between the taxa is. In other cases it may suffice to display that an evolutionary event *has* taken place, without giving any further information. In the following, though, the edge lengths will be regarded as an important value. Their calculation may differ strongly from algorithm to algorithm: in the UPGMA method they result quite directly from the distance between the taxa connected by that edge, whereas the BME/BNNI algorithms demand a more complex computation.

In this section, these three algorithms will be described, and finally some details on their implementations will be presented. The notation used in the following for describing phylogenetic trees and their distance matrices has been adopted from Desper and Gascuel [4].

2.1 *The UPGMA Algorithm*

The algorithm's name, *Unweighted Pair Group Method with Arithmetic Mean*, harbors its operating principles, which will be explained in the following.

The algorithm works by *clustering*, iteratively joining the two clusters with the smallest distance. To begin with, every taxon can be seen as its own cluster; the distances between the clusters correspond to the distance matrix. As a first step, the two clusters with the smallest distance, i.e. the smallest evolutionary change, are merged to form a joint cluster, by introducing a new node to connect them. This new node is now the cluster's root, and it is the one that will be connected with another cluster's node in a future merging step. This process of searching for the best merge—and performing it—is repeated until only one cluster, containing all taxa, remains. The node that is created by merging the two final clusters becomes the tree's root. The produced tree is always a binary tree, whereas every node has exactly two children or none.

This brings us to the required input, namely a *distance matrix* Δ which specifies the evolutionary distance between the different taxa. This matrix should ideally be ultrametric, and should thus meet several conditions [7]:

First of all, the matrix should be *symmetric*, meaning that the distance of taxon a to taxon b equals the distance of b to a. Thus, for a $n \times n$-matrix the following holds:

$$\Delta_{ab} = \Delta_{ba} \quad \text{for all} \quad a, b \in \{1, ..., n\}. \tag{1}$$

Secondly, the matrix should fulfill the *triangle inequality*, i.e.

$$\text{for all} \quad a, b, c \in \{1, ..., n\} \quad \text{we have} \quad \Delta_{ac} \leq \Delta_{ab} + \Delta_{bc}\ . \tag{2}$$

This states that the distances are to be "true" distances, i.e. Euclidean distances, such that the direct way from point a to c cannot be longer than a detour over point b.

Thirdly, the distance of a taxon to itself is zero, and all other distances are larger than zero:

$$\text{for all} \quad a, b \in \{1, ..., n\} \quad \text{we have} \quad \Delta_{ab} = \begin{cases} 0, & \text{if } a = b, \\ x, & \text{if } a \neq b;\ x > 0. \end{cases} \tag{3}$$

Equations 1, 2, and 3 are the conditions for a metric. As the UPGMA algorithm works on the assumption that all taxa evolve at the same pace, resulting in identical distances of all taxa to the root, one more condition should ideally be fulfilled, making the distance matrix ultrametric: The *three-point condition* states, that when looking at three taxa, two of the pairwise distances must be equal to each other, while the third distance is either smaller or also equal to them; any three taxa $a, b, c \in \{1, ..., n\}$ can be named such that we have

$$\Delta_{ab} = \Delta_{ac} \geq \Delta_{bc}\ . \tag{4}$$

Now that it has been clarified what kind of an input matrix is required, we can take a look at how the distance between two clusters is calculated on the basis of that matrix [7]: The distance of two clusters C_i and C_j to each other is defined as follows, using *arithmetic means*:

$$dist(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{\substack{a \in C_i \\ b \in C_j}} \Delta_{ab} \,. \tag{5}$$

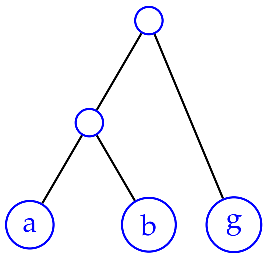
The distance of two clusters is thus simply the average of all pairwise distances between the taxa in C_i and C_j. As the distances are all weighted equally in the calculation, regardless of cluster sizes and of the taxa's positions in the tree, UPGMA is said to operate in an *unweighted* way[1]. For the actual implementation of the algorithm, it would be impractical to rely solely on Equation 5. With the help of an update formula, the distance between two clusters does not have to be calculated from scratch after each merging step: When two clusters C_i and C_j are merged to $C_{i,j}$, the new cluster's distance to any third cluster C_k can be calculated from its previous cluster distances to C_i and C_j, again by using arithmetic means:

$$dist(C_{i,j}, C_k) = \frac{dist(C_i, C_k)|C_i| + dist(C_j, C_k)|C_j|}{|C_i| + |C_j|} \,. \tag{6}$$

This saves run time and is easy to implement.

Although the distance matrix should ideally fulfill the requirements described beforehand, it is not mandatory, and the algorithm will produce a tree in any case. The quality of the tree, however, may be affected negatively. The distance matrix is only fully represented by the tree if the matrix is ultrametric. In order to demonstrate how the properties of the matrix affect the tree quality, three example matrices have been created, as displayed in Figure 2.2. The trees created by UPGMA upon processing the respective matrices can be seen in figure 2.1.

While the tree produced with matrix 2.2(a) represents the distances perfectly, i.e. the edge lengths mirror the distances, the tree constructed on the basis of matrix 2.2(b) for example fails to distinguish between Δ_{ac} and Δ_{ad}, simply averaging their values. Matrix 2.2(c) does not only violate the three-point condition, but also the triangle inequality. The information that the pairs of taxa (a, b) and (a, g) are close, while (b, g) have a large distance, is not displayed in the tree; see also the tree on the right. In fact, this cannot be displayed by any tree (in the Euclidean space). The UPGMA algorithm instead produces a tree that displays *either* the closeness of (a, b) *or* that of (a, g) at the expense of the other pair (in this case the closeness of (a, b) is depicted). In Figure 2.1(c) it can be seen that the leaf g moved closer to a and thus also closer to b, as a result of the small values of Δ_{ag}. This happens at the expense of the accuracy of the distances of g to all other taxa, including Δ_{bg}, which is actually much larger than displayed in the tree.



[1]The *WPGMA* algorithm can be seen as UPGMA's weighted counterpart [4]; they differ in the calculation of the cluster distances. WPGMA will not be discussed any further in this thesis.

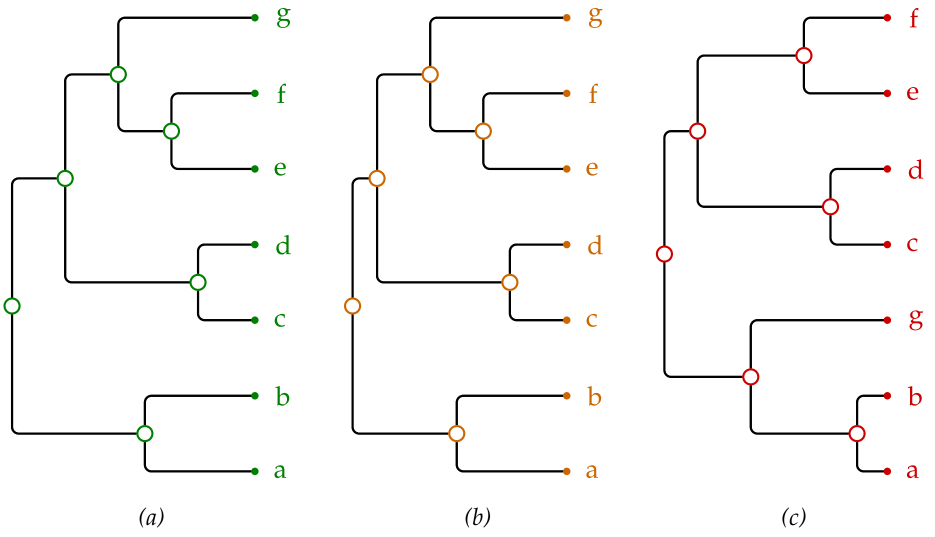


Figure 2.1: The trees produced by UPGMA upon input of the matrices displayed in Figure 2.2. *(a):* The tree that is produced when the ultrametric distance matrix 2.2(a) is used as input. *(b):* The tree that is produced when the distance matrix 2.2(b), which does not fulfill the three-point condition, is inputted. *(c):* The tree generated when the matrix 2.2(c), which neither fulfills the three-point condition nor the triangle inequality, is used as input.

	a	*b*	*c*	*d*	*e*	*f*	*g*
a	0	4	9	9	9	9	9
b	4	0	9	9	9	9	9
c	9	9	0	2	7	7	7
d	9	9	2	0	7	7	7
e	9	9	7	7	0	3	5
f	9	9	7	7	3	0	5
g	9	9	7	7	5	5	0

(a)

	a	*b*	*c*	*d*	*e*	*f*	*g*
a	0	4	9	**7**	9	**6**	9
b	4	0	9	**7**	9	**6**	9
c	9	9	0	2	7	7	7
d	**7**	**7**	2	0	7	7	7
e	9	9	7	7	0	3	5
f	**6**	**6**	7	7	3	0	5
g	9	9	7	7	5	5	0

(b)

	a	*b*	*c*	*d*	*e*	*f*	*g*
a	0	**1**	9	9	9	9	**1**
b	**1**	0	9	9	9	9	9
c	9	9	0	2	7	7	7
d	9	9	2	0	7	7	7
e	9	9	7	7	0	3	5
f	9	9	7	7	3	0	5
g	**1**	9	7	7	5	5	0

(c)

Figure 2.2: Example matrices with seven fictional taxa. Differences of (b) and (c) in comparison to (a) are marked red. *(a):* An ultrametric distance matrix. *(b):* A distance matrix with unfulfilled three-point condition. *(c):* A distance matrix with unfulfilled triangle inequality.

2.2 *The BME Algorithm*

The *balanced minimum evolution* algorithm, as described by Desper and Gascuel [4], builds a phylogenetic tree on the assumption of *minimum evolution*, i.e. it searches for the tree that explains the given data with as few evolutionary events as possible. In contrast to UPGMA, the BME algorithm does not work by clustering, but instead it iteratively adds one leaf after another to the current tree. Hereafter the algorithm will first be outlined, then some important definitions and details will be given, and finally some examples will be considered.

The algorithm builds the tree by adding one leaf after another, until all leaves have been included in the tree. To make things easier, the taxa are numbered from $k = 1$ to n. The tree is initialized by connecting the first three leaves via a new inner node. Before the next leaf ($k = 4$) is inserted, all average distances of k to any subtree in the current tree are computed. Then any edge in the tree T_{k-1}, i.e. the tree with $(k-1)$ leaves, is considered as a possible insertion point for leaf k and the resulting total *tree length*, a term that will be explained a little further down, is determined. As we are looking for a tree that explains the data on an assumption of *minimum evolution*, k is finally inserted into the edge that minimizes the tree length: For this purpose a new node is created, connecting k and the two nodes which were previously direct neighbors. Finally, all average distances influenced by k need to be updated. These three steps—calculating the distances to k, finding the best edge and inserting k there, and updating the distances—are repeated, until all leaves have been added to the tree. Finally, the true edge lengths must be computed or the subtree-swapping algorithm BNNI must be called.

As described above, BME depends on the computation of the average evolutionary distances between subtrees. This is done by using the pairwise distances of the taxa described in the distance matrix. The average distance between two subtrees A and B in a topology τ is defined as follows:

$$\Delta^{\tau}_{A|B} = \begin{cases} \Delta_{ab}, & \text{if } A = \{a\} \text{ and } B = \{b\}, \\ \frac{1}{2}\left(\Delta^{\tau}_{A|B_1} + \Delta^{\tau}_{A|B_2}\right), & \text{if } B = B_1 \cup B_2, \end{cases} \tag{7}$$

with B_1 and B_2 being the two non-intersecting subtrees of B, as shown in Figure 2.3. As the τ in $\Delta^{\tau}_{A|B}$ suggests, in BME, the distance between two subtrees is dependent on the topology. This means that not only the question of *which taxa are part of the subtree* matters, but also the question of the *arrangement of the taxa within the subtree*. The (final) edge lengths need to be calculated explicitly; they are not computed on the fly as in UPGMA. The internal edge e in the model tree (see Figure 2.3) for example is computed by calculating the distance between the subtrees $A \cup B$ and $C \cup D$. But since we only want the length of e, the distances $\Delta^{\tau}_{A|B}$ and $\Delta^{\tau}_{C|D}$, which were added obligatorily, are subtracted again:

$$l(e) = \Delta^{\tau}_{A\cup B|C\cup D} - \frac{1}{2}\left(\Delta^{\tau}_{A|B} + \Delta^{\tau}_{C|D}\right). \tag{8}$$

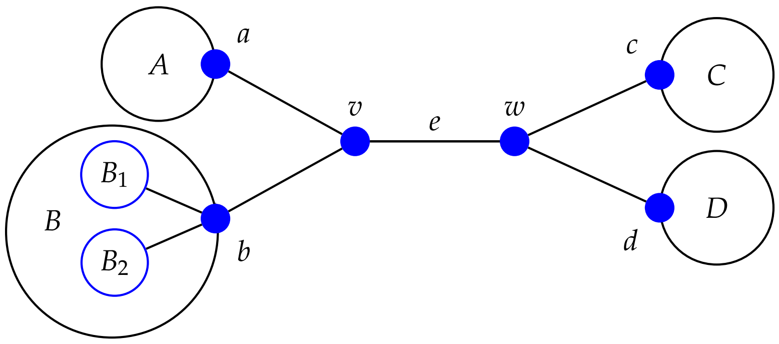


Figure 2.3: A tree with labeled edges and nodes, in consensus with the BME/BNNI notation, as shown similarly by Desper and Gascuel in [4].

The calculation of external edges can be performed as follows, with i as the respective leaf:

$$l(e) = \frac{1}{2}\left(\Delta^{\tau}_{i|A} + \Delta^{\tau}_{i|B} - \Delta^{\tau}_{A|B}\right). \quad (9)$$

The *tree length* is a term which takes on large values if leaves with a large evolutionary distance are close neighbors in the topology. In other words, the tree length validates the tree, becoming minimal when the distance matrix is reflected in the topology. It can be calculated by simply adding up all edge lengths.

Like UPGMA, BME also requires its input distance matrix to ideally fulfill the symmetry statement, the triangle inequality, and the identity statement. The three-point condition plays no role, as BME does not assume a constant mutation rate, allowing the different taxa to evolve at different speeds. Thus the taxa do not necessarily have the same distance to their common ancestor. In fact, BME does not make any suggestion about such a common ancestor and does not produce a rooted tree. This means that BME does not indicate the direction or order of the evolutionary events.

For the actual implementation of BME, it would be very inconvenient to determine the complete tree length before every leaf-insertion step, especially with regard to the computation time; there is an update formula that can be used instead. This formula does not directly compute the tree length, but instead merely determines the *change in tree length* from one topology to the next. Similarly, the updating of the pairwise subtree distances after each leaf-insertion step does not have to be done from scratch, but instead the preceding distances may be used. The number of distances to be updated here should not be underestimated: the only distances that do *not* need to be updated are distances between two subtrees of which neither contains the newly inserted leaf k; for the actual formulas see [4].

Figure 2.4 shows the trees the BME algorithm produces when the matrices displayed in Figure 2.2 are used as input. The tree 2.4(a) perfectly mirrors its ultrametric distance matrix. In fact, the branch lengths correspond exactly to the respective ones in tree 2.1(a), produced by UPGMA. The tree 2.4(b) displays the differences for example between Δ_{ac} and Δ_{ad} and

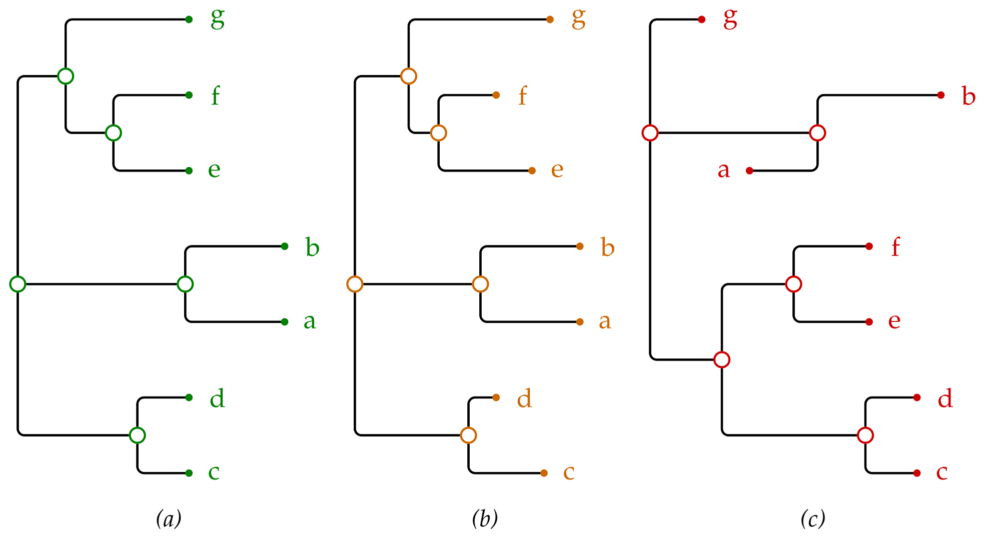


Figure 2.4: Trees produced by BME on the basis of the distance matrices displayed in Fig. 2.2. Note that the algorithm produces an *unrooted* tree; a graph drawing algorithm introduced a root afterwards. *(a):* The tree produced on the basis of matrix 2.2(a). *(b):* The tree produced on the basis of matrix 2.2(b). *(c):* The tree produced on the basis of matrix 2.2(c).

also between Δ_{eg} and Δ_{fg}, unlike its UPGMA counterpart. The *real* distance between the taxa pairs (e, g) and (f, g) is 5, as defined by the distance matrix. In a tree that perfectly mirrors the matrix, adding up all branch lengths between two of those pairs should result in that value; this is the case for tree 2.4(a) and thus we have

$$l(e, g) = l(f, g) = 5 = \Delta_{e,g} = \Delta_{f,g},$$

with $l(e, g)$ being the sum of the edge length between e and g. Since, among others, in tree 2.4(b) the distances Δ_{ae} and Δ_{af} are *not* equal, the tree has to compromise by for example adjusting the lengths of the leaves e and f to their common root. This results in slight deviations from their real distances to g:

$$\Delta_{e,g} \neq 5.29 = l(e, g) \neq l(e, f) = 4.64 \neq \Delta_{e,f}.$$

A tree computed on the basis of matrix 2.2(c) has to deal with non-Euclidean distances, as described before. This results in a negative branch length, which, mathematically is a solution, but makes no sense biologically, as a negative evolutionary distance is undefined:

$$l(a, b) = 1 = \Delta_{ab}, \quad \text{but:} \quad l(a, r) = -1.5 \quad \text{and} \quad l(b, r) = 2.5,$$

with r being the node connecting a and b.

2.3 The BNNI Algorithm

In contrast to BME and UPGMA, the *balanced nearest neighbor interchange* algorithm, also presented by Desper and Gascuel [4], does not build a tree from scratch, but instead enhances an already completed tree. The BNNI is a heuristic for minimizing the tree length; it operates by swapping subtrees of the completed tree and renders a possibility of improving the results of BME or also other distance-based algorithms.

One of the input requirements for the BNNI is thus a complete tree T; the distances between all non-intersecting subtrees are another requirement. If BNNI is applied subsequently to BME these distances have already been computed; otherwise they need to be calculated on the basis of the respective distance matrix, with help of Equation (7).

The BNNI algorithm operates by iterating over all internal edges of the tree and looking for the best subtree swap, whereas only subtrees separated by three edges are considered as swapping partners. When regarding an internal edge e (see Figure 2.3) there are three possibilities of how the subtrees A, B, C, and D can be positioned relatively to each other. The algorithm runs over all three possibilities and compares their effect on the tree length. The information on the swap that would lead to the strongest decrease in tree length is saved, if such a swap exists. The change in tree length can, once more, be calculated with an update formula, which saves run time.

Of all tree-length-improving swaps the overall best swap is performed, whereupon the distance matrix needs to be updated (again with the help of different update formulas, see [4]). These two steps are repeated until no such swap is left, improving the tree length every time. Finally, the actual branch lengths can be calculated. For this the precomputed matrix distances are used.

Since the BME already aims at producing an optimal tree with respect to the *minimal evolution* criterion, small trees are likely to already be in a (local) minimum after the BME algorithm is finished. Thus, for BNNI to have an actual effect, the input has to be of a certain size. The sample trees displayed so far are all so small that no optimizing swaps are found (see Section 2.4 for an example with a larger input matrix). As stated by Desper and Gascuel [4], the BNNI algorithm performs approximately one swap if the input comprises 24 taxa; 96 taxa yield about seven swaps, a tree with 4000 taxa undergoes over 116 swaps on average.

The book *Phylogenetic Networks* [7] praises the BME algorithm for the fact that it *does not produce negative edge lengths*, in comparison for example with the neighbor-joining method. Readers who had a careful look at Figure 2.4 will be disconcerted: the tree grows right in all three cases, but there is definitely an edge that grows left, meaning is has a *negative* edge length. A further paper by Desper and Gascuel [5] gives insight into when exactly the statement—that no negative edge lengths are produced—holds: The tree has to be in a local minimum concerning the tree length, which it is if BNNI is applied. Under that condition, no internal edge will have a negative length, as proven in the paper mentioned above. This holds even if the distance matrix does not fulfill the triangle inequality. External edges, on the other hand, may be negative if the distances are no Euclidean distances, as it is the case for the tree in Figure 2.4(c).

2.4 *Implementation—Details, Difficulties, and Evaluation*

This section deals with the implementation of the algorithms described above. To begin with, the overall structure of the module will be outlined, in order to demonstrate which part of the module we are currently dealing with. Subsequently some details and difficulties of the implementation of the phylogenetic tree algorithms will be discussed, followed by an evaluation of the implementation.

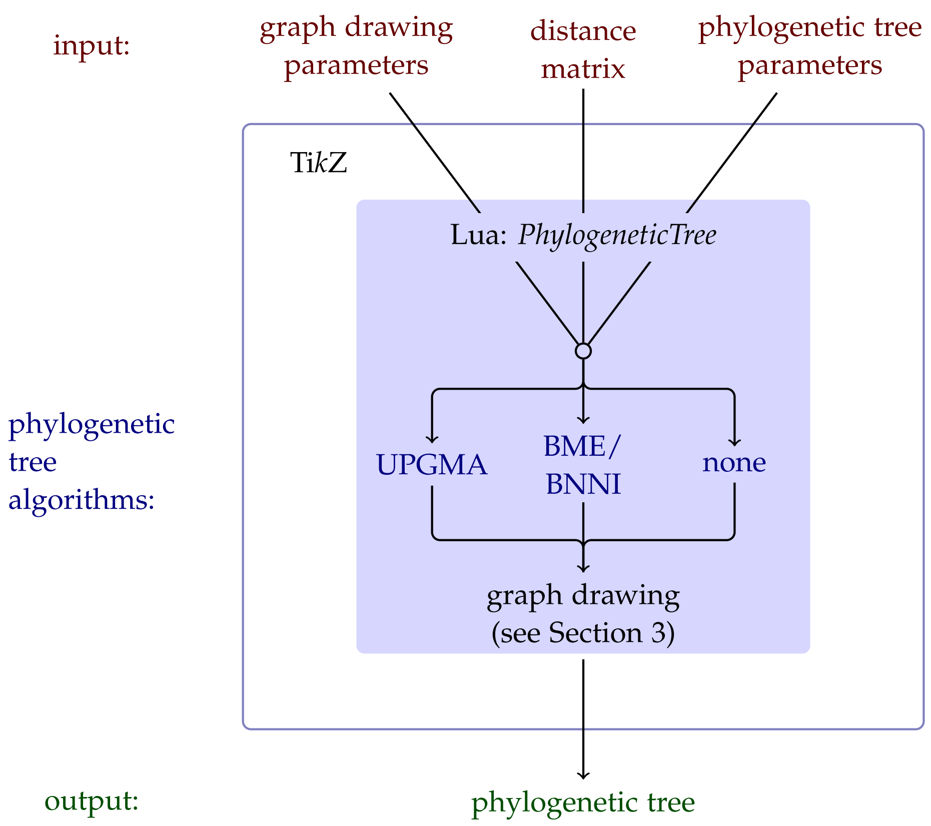


Figure 2.5: A diagram roughly depicting the overall structure of the implemented module. The user inputs a distance matrix and parameters, by typing TikZ code in their TEX document. This information is handed to the *PhylogeneticTree* class, which is part of the Lua layer. According to the user's specifications, either UPGMA, BME/BNNI or no phylogenetic tree algorithm is called. In the latter case, the users have to specify the tree's topology themselves. Next, a graph drawing algorithm is evoked; again dependent on the user's input. Lua hands the graph information back to TikZ and finally a completed phylogenetic tree is rendered.

As can be seen in Figure 2.5, the phylogenetic algorithms are handled by the class *PhylogeneticTree*, which analyzes the user's input and then invokes the accordant subclasses handling the phylogenetic tree and graph drawing algorithms. For an example of how to specify the required input, see Figure 1.3; a full list of the parameters is given in Appendix B.

Programming in *Lua* means programming by using *tables*, the language's most important—or rather only—data structure. Tables occupy the role of *classes*[2] as well as of *objects* in object-oriented programming; they can contain any kind of entry, also functions. There are several classes used to model graphs on the Lua layer in Ti*k*Z, such as the *Digraph* and the *Vertex* classes, which were created by Till Tantau [15]. These classes have also been used for programming the phylogenetic trees module.

The implementation of the UPGMA algorithm was quite straightforward. It was especially useful to me for training to program in Lua and to become familiar with the graph model classes. In addition to the already existing ones, another class, the *Cluster*, was needed and therefore implemented: A cluster object saves the information on the nodes it contains and its distances to other clusters, which are needed for computing the next merging step, as described in Section 2.1. Analogously to vertices and arcs, the clusters can be stored in the digraph object, which is accessible via phy_tree.digraph in the UPGMA class. In Lua this is an easy-to-do modification, as we can simply create a new table in the existing digraph object within the UPGMA class:

```
self.phy_tree.digraph.clusters = {}
```

There is no need to create an actual matrix to represent the distance matrix, which in Lua would be implemented as a table of tables. Instead, the newly computed pairwise distances of the clusters to each other are stored directly in the respective cluster objects.

The BME/BNNI algorithms, on the other hand, were far less trivial to implement. This is, for one, due to the algorithms' general concepts, which are much more complex than UPGMA's: alone the process of understanding BME and BNNI was time consuming. In the actual implementation, the main difficulties lay with the many updating steps: they had to be carefully implemented in order to ensure that every distance that needs to be updated *is* indeed really updated—with the correct value.

The distance matrix is handled slightly different than with UPGMA: newly computed distances are stored in the vertices' storages, which are objects designed especially to store any kind of information [15]; again there is no need to create an actual matrix.

As computing the distances between subtrees is a central part of the BME algorithm, a question that arose very early was how to distinguish between the different subtrees, i.e. how to identify them unambiguously: A subtree can be defined by its root, and the distance between two non-intersecting

[2]Lua does not explicitly offer a class system. Classes may be imitated by making an object the *prototype* of another by making it the other's *metatable*[8].

subtrees can be saved unambiguously by saving it as the distance between the two roots. In Figure 2.3, for example, the distance between subtrees B and $C \cup D$ can be stored as the distance between the two nodes b and w. The distance between $A \cup B \cup C$ and D, on the other hand, can be stored by using w and d as references. Thus, to which subtree the node w refers to depends on the subtree we are comparing it to. Analogously, when regarding two leaves, the distance can be saved as the distance between these two nodes.

Less clear was how to implement the step that computes all distances of the to-be-inserted leaf k to all non-intersecting subtrees; there was no solution offered by Desper and Gascuel in [4]. As k has, in that step, not yet been inserted and the distances to *all* subtrees are required, every inner node must describe three different subtrees. This can be done by remembering the direction from which we are looking at a vertex. Considering Figure 2.3 once more, one can see that w acts as the root of subtree $C \cup D$ if one regards it from perspective of the vertex v. Thus, using v as the center, one can see the three subtrees w, b (as the root of B) and a (as the root of A). This approach has been implemented by storing the distances of k to the subtrees in a temporary table of tables:

```
local k_dists = {}
k_dists[center_vertex] = {}
k_dists[center_vertex][root] = distance
```

The variable *center_vertex* contains the vertex we are currently using as the "center" and the variable *root* is the vertex functioning as a subtree's root. The parameter *distance* contains the computed distance, done so by applying Equation (7) recursively [4]. Once k has been inserted into the tree, the distances that are still relevant are incorporated into the distance matrix; the other distances are discarded.

Implementing the updating steps is very error-prone; especially time consuming was the *detection* and then the *correction* of implementation mistakes, especially those yielding no error report. To name just one example, in the step after the insertion of taxon k into the tree, where all distances of k to still relevant subtrees are incorporated into the distance matrix, I accidentally introduced such a mistake: The tree produced by the algorithm looked almost perfect, but by comparing the tree in Figure 2.4(a) and its respective distance matrix, it became apparent that for example the edge lengths of leaves a and b to their common parent must be identical. This was not the case, and as the source of that error could have been almost *anywhere*, a systematic search began—until the mistake was finally eliminated successfully. Such error findings made clear that the implementation of the algorithms should be carefully scrutinized, which will be done a little further down.

During the implementation of the BNNI algorithm, the updating steps were again the most critical steps. Additionally, a *heap* was needed to store the possible swaps efficiently, a data structure I had not worked with before. The heap was implemented as a binary heap, which turned out to be quite straightforward.

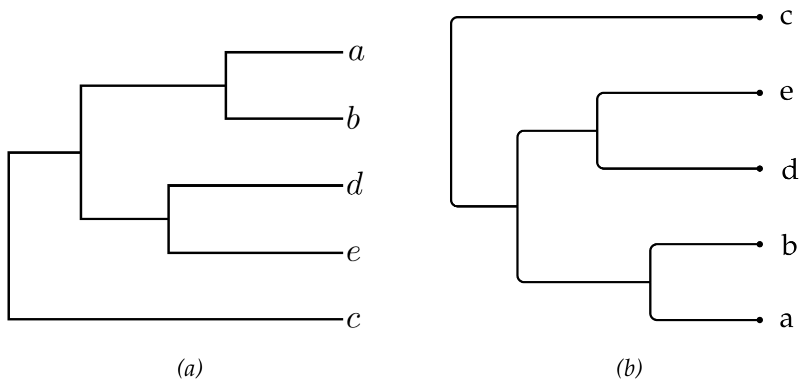
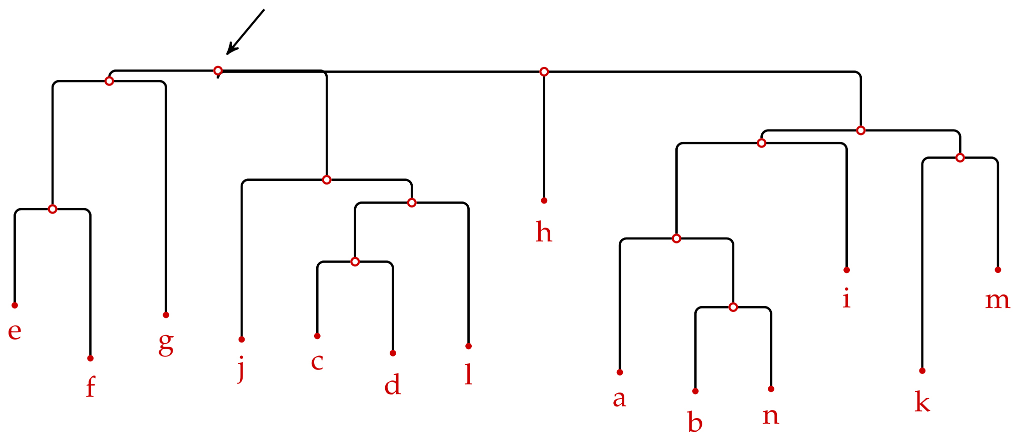


Figure 2.6: (a): The tree displayed in Figure 3.20(b) in *Phylogenetic Networks* [7], computed on the basis of the distance matrix depicted in Figure 3.20(a) in [7]. *(b):* The tree computed by the implemented UPGMA algorithm, when the same distance matrix is used.

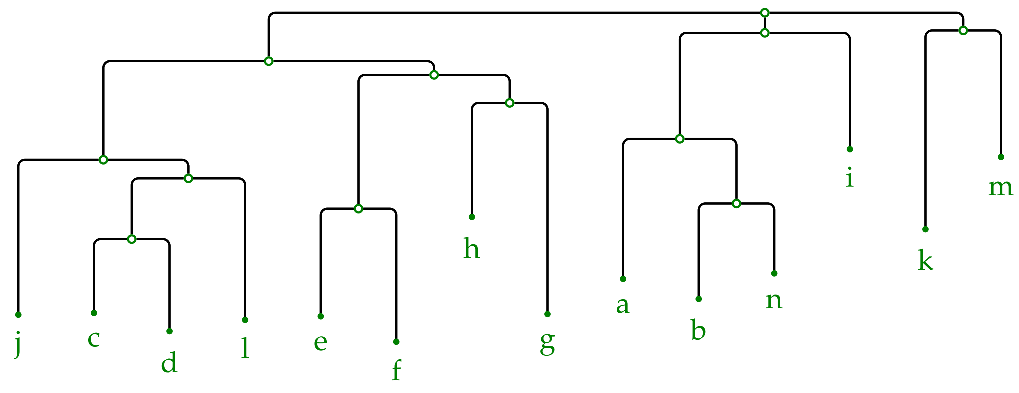
Let us now turn to the evaluation of the implementations. To verify that an algorithm has been implemented correctly is not trivial. In the following an attempt at *evaluating* the algorithms' correctness will be made, although *no* actual *proof* will be given.

Whether the UPGMA algorithm has been implemented correctly can be evaluated by having a tree computed on the basis of the distance matrix displayed in *Phylogenetic Networks* ([7]: p. 53, Figure 3.20), as suggested in *Exercise 3.13.1* on the same page. This has been done; the results are depicted in Figure 2.6: The tree produced has the same topology as the one in the book. Furthermore, the edge lengths seem to have the same ratio to each other, although there are no numeric data given in the book to confirm this observation and the book does not clarify whether the taxa labels are to be seen as part of the edge or not. Even though the assumption that the algorithm has been implemented correctly is not proven by this, it is at least supported by this result.

In order to try to evaluate whether the implementation of the BME algorithm is error-free, two computed results (the trees in Fig. 2.4(a) and 2.4(c)) have been validated by calculating parts of their final distance matrices *without* using the results of the updating formulas, but instead by recursively using Equation 7, which *defines* the distances between subtrees. If the matrix is correct, so are the edge lengths, as they can be simply computed by solving Equations 8 and 9. This test run demonstrated that the calculation of the edge lengths and its underlying distances is performed correctly—at least for the tested examples; it did not, though, directly test whether the tree is built correctly. But, if the distance matrix is correct, it is unlikely that the tree is erroneous, as the construction of the tree is directly dependent on the matrix. If BNNI is run subsequently and no subtree swapping is performed, it can be gathered that a local tree length minimum was found by BME, making an incorrect BME implementation even more unlikely.



(a)



(b)

Figure 2.7: Trees with 14 Taxa computed on the basis of the same distance matrix (see Figure C.1 in Appendix C). *(a):* The tree produced by BME, without the BNNI algorithm performing any branch swapping. A slightly negative edge length is marked by an arrow. *(b):* The tree produced by BME with subsequent optimization by BNNI in the form of two swaps.

This brings us to the inspection of the BNNI implementation—which is challenging because of the large input sizes it requires, as stated in Section 2.3. During the programming phase, the condition for carrying out a swap was changed, so that a specific number of swaps were performed instead of just those that improve the tree length. This, of course, resulted in poorer results, but enabled the testing of the code. To assess whether the BNNI implementation really functions correctly and *improves* the tree length, the usage of a larger exemplary input matrix cannot be evaded. The trees in Figure 2.7 were produced on the basis of a distance matrix describing 14 taxa. It should be noted that the underlying distance matrix has been constructed with random numbers and does *not* fulfill the triangle inequality for many pairs of taxa. Figure 2.7(a) shows the tree produced by BME without BNNI, while Figure 2.7(b) depicts the same tree after BNNI performed two branch swaps. There is one slightly negative internal edge length in tree 2.7(a), marked with an arrow, whereas in tree 2.7(b) all edge lengths are positive—which is a clear improvement of the tree and, as stated earlier, negative internal edges can only occur if the tree is not in a local tree length minimum. Although this test run does, of course, not *prove* that the algorithm functions correctly, the fact that the tree length has been improved by BNNI supports this assumption: While tree 2.7(a) has a total length of 29.80, the length of the tree in Fig. 2.7(b) is only 29.37.

All in all, the test runs suggest that the implementation of the algorithms are at least free of major errors. The performance is fast; the examples depicted in this thesis are each computed in less than a second: For the largest examples computed by BME (Figure 2.7) the run time lies at about 0.12 s, the graph drawing for the same trees takes approximately 1 ms. The branch swapping by BNNI takes about 1 ms for Figure 2.7(b); UPGMA takes at most 0.06 s to compute the given examples.

3 VISUALIZING PHYLOGENETIC TREES IN TEX

Once the topology of the tree has been defined for good, whether by a phylogenetic algorithm or by the user, the tree still has to be *visualized*. The essence of the visualization process is to set the positions of the nodes in the plane, i.e. their x- and y-coordinates. *Decorative* visualizing elements, such as the color of the edges or the shape of the nodes, are handled by Ti*k*Z at a different level; it merely has to be made sure that the user has the possibility of adjusting those decorative elements.

Though other graph drawing algorithms have already been implemented as part of the Ti*k*Z graph drawing library, none of them allow the option of *fixed edge lengths*. As exactly this property is *the* important factor for the type of phylogenetic trees dealt with in this thesis, a new graph drawing algorithm has been implemented. Such trees with edge lengths that are proportional to evolutionary change are often called *phylograms* [7].

Some phylogentic algorithms, like UPGMA, always produce rooted trees: Every inner node is either of degree three, or is the root and thus of degree two. In unrooted trees produced by other algorithms, as for example BME, all inner nodes are of degree three. There may be other settings for trees computed by other algorithms, as for example inner nodes of degree two or four, thus the graph drawing algorithm has been designed to allow all kinds of degrees.

This section deals with the graph drawing algorithm—designed by myself—that has been implemented as part of the module. This graph drawing algorithm offers several different layout options, which will be explained and discussed hereafter; the basal layout can be specified by the user via the key *phylogenetic layout* (see Figure 3.1). The first two parts to follow are about drawing rooted and unrooted trees, respectively—the two most important layout options. The last part deals with some further options, which are applying straight edges instead of rectangular ones, using a scaling factor to control the tree size, and changing the style of the inner nodes. Additionally, the edge length will be defined and the effect of certain rotating options on the edge lengths will be discussed.

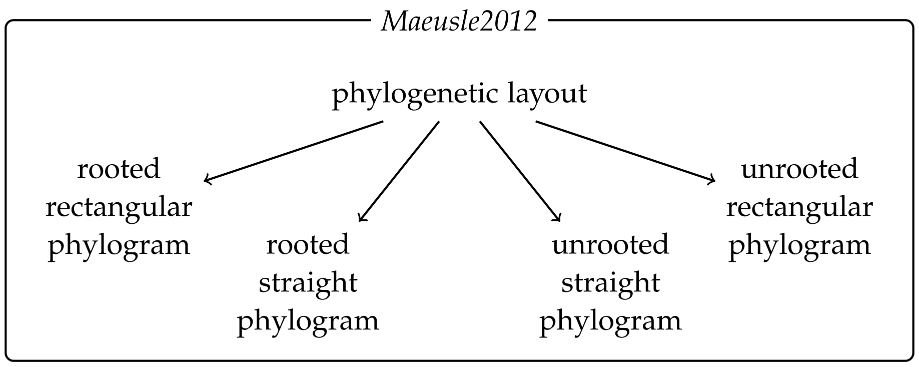

Maeusle2012
phylogenetic layout
rooted rectangular phylogram
rooted straight phylogram
unrooted straight phylogram
unrooted rectangular phylogram

Figure 3.1: The implemented graph drawing class, *Maeusle2012*, has four main layout options, as depicted above. The edge lengths of the resulting trees are always proportional to the amount of evolutionary change as defined by the distance matrix; the trees can thus be referred to as *phylograms* [7].

3.1 *Drawing Rooted Trees*

In this section we will look at how a rooted rectangular phylogram[3] is built; information on phylograms with straight edges will be added in Section 3.3. A rooted rectangular phylogram has its edges bent in 90° angles, see Figure 3.2 for an example. The implemented algorithm can be summarized as follows:

1. Look for nodes of degree two.
   (a) If at least one such node exists, choose the first one as root of the tree,
   (b) else compute the center of the tree and use the most central node as root.
2. Beginning at the root, use a depth-first traversal to set the x- and y-positions.

Let us take a closer look at the individual steps. As the algorithm sets the positions for a *rooted* tree (see Figure 3.2 for illustration of the used coordinate system), the first step consists of determining the tree's root. Step 1(a) is quite straightforward. As an inner node embodies an evolutionary event, which generally results in at least two different lineages, it usually makes no sense for phylogenetic trees to have more than one node of degree two. Nevertheless, if there are more nodes of degree two, a smarter choice of root could be implemented, such that the most central of those nodes is made root, similarly to step 1(b): A balanced tree tends to be more appealing to the eye than a tree with very differently sized subtrees; it is usally also more compact and thus increases its likelihood of fitting on the page of a document (see for example [1] on graph aesthetics).

[3]An algorithm similar to mine is described in [7], a fact I only noticed after my implementation was finished. The name *rectangular phylogram* was adopted from [7].

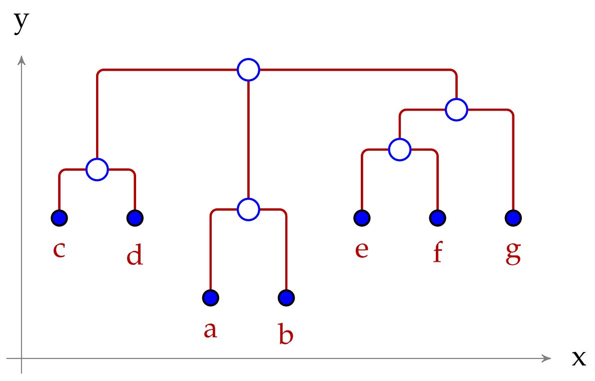


Figure 3.2: The definition of the x- and y-coordinates of a tree used in this thesis. The growth direction is set to *down* and the layout is specified as *rectangular phylogram*, which results in the rooted style with bent edges. The tree was computed by BME whereas the matrix in Figure 2.2(a) was used as input.

In the aforesaid step the root is chosen by determining the tree's most central node. For this, the longest path in the tree is identified. This can be done as follows[4]:

i. Choose any node as the starting point.
ii. Compute the path lengths from that node to all leaves, by adding up the lengths of the edges on the respective paths.
iii. Take the leaf of the longest path from the previous step and repeat step ii with that node as the starting point.
iv. Choose the longest resulting path; this is the tree's overall longest path.

That this really is the tree's overall longest path can be concluded from the following thoughts: We begin with a node anywhere in the tree. The longest path from that node to any other node will always lead to a leaf. That leaf must be the starting or end point of the overall longest path, otherwise we would not have gotten there. Beginning at that leaf, the search for the longest path will provide us with the globally longest path.

If there is a node directly in the middle of the longest path, the choice of the root is straightforward: that one will be taken. Otherwise the node closest to the center is calculated and chosen as the root.

Once the root has been determined, step 2 can begin: the actual positioning of the nodes. Using a depth-first search, the y-positions of the leaves are set to the path length from the root to every individual leaf. This means that the edge length is displayed only by the height between nodes, and not by their distance along the x-axis. This decoupling of x- and y-positions facilitates the calculation of the positions on the one hand, and renders an easy possibility for the viewer of a tree to estimate the distance between two nodes on the other hand.

[4]My thanks to Christoph Stockhusen for this approach.

The first leaf's x-position remains 0; the next leaf's x-position is set to the "ideal sibling distance" [15], which takes the size of the nodes into consideration, so that they do not intersect. Inner nodes are placed in the middle of their children: if the number of children n is even, the parent's x-position is set to $(x_1 + x_n)/2$. In case an inner node has an odd number of children, it is placed above the middle child. In accordance with this, a parent node with only one child—which usually makes no sense when dealing with phylogentic trees—get assigned the same x-position as its child. The y-position is in all cases dependent on the pre-calculated edge lengths. Finally, the positions of the root are set; its y-position remains zero.

3.2 *Drawing Unrooted Trees*

In some cases the display of a tree with a root may be misleading: the BME algorithm for example produces an unrooted tree and does not necessarily suggest a common ancestor to the taxa. Thus, the depiction of the tree in an *unrooted layout* should be—and is—an option. The algorithm for this works by dividing the tree into two subtrees and setting the positions for them separately, whereby the two subtrees are also connected by an edge of the correct length. When using the same input as for the tree in Figure 3.2, but with the layout option set to *unrooted rectangular phylogram*, the tree displayed in Figure 3.3(a) is produced.

The algorithm operates as follows:

1. Determine the root as in Section3.1, but additionally determine the roots nearest neighbor.
2. Taking these two nodes as the starting points, use a depth-first search for each to set the positions for the respective subtree.
3. If necessary, shift the x-positions of one tree half for a more aesthetic result.

The first step is quite straightforward. The root and its nearest neighbor can be seen as the roots of two non-intersecting subtrees, which together make up the complete tree. In step 2 the nodes' positions are set similarly to the way described in the previous section, with the difference that the two subtrees are handled separately. The growth direction of the two subtrees are opposite to each other, see Figure 3.3(a) for an example: The node marked with an asterisk is the computed root and the node marked with two asterisks is its nearest neighbor.

Finally, in step 3, the x-positions of the two roots are compared—if they are not the same, the x-positions of one subtree are shifted by their difference, to make the figure more aesthetic.

3.3 *Applying Further Options: Straight Edges, Scaling Factors, and Inner Node Styles*

Here, first of all the usage of straight edges instead of rectangular ones will be discussed as additional layout options. We will also look into the problem

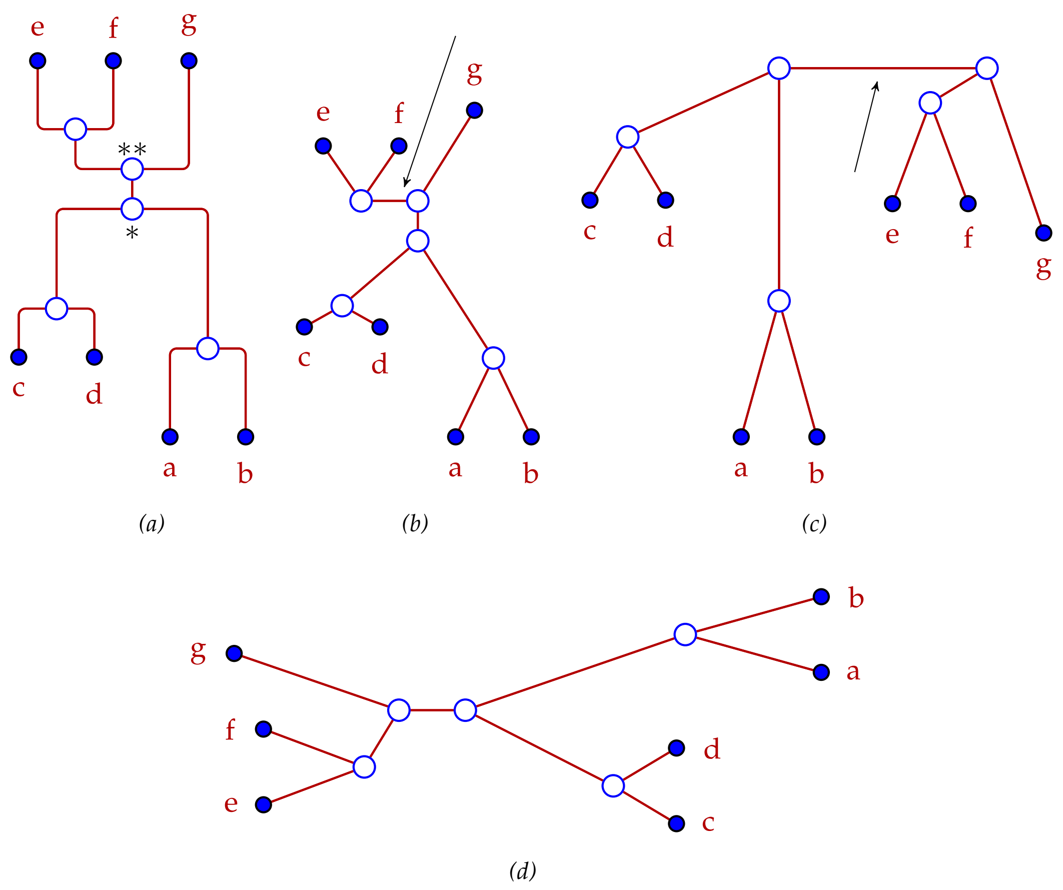


Figure 3.3: Trees with different layout options produced by BME with Fig. 2.2(a) serving as distance matrix. *(a):* The tree produced when the layout option is set to *unrooted rectangular phylogram*. *(b):* The resulting tree when the layout option is set to *unrooted straight phylogram*. *(c):* The tree produced when the layout option is specified as *straight phylogram*. *(d):* The resulting tree when the layout option is set to *unrooted straight phylogram*, together with a higher distance scaling factor.

of scaling a phylogenetic tree correctly, and then turn to the question of how the inner nodes' style can be customized. In order to enable the correct interpretation of the trees, a definition of the edge length will be given, and finally, the effect of rotating parts of the tree on the correct edge lengths will be analyzed.

Although it is not uncommon to display phylogenetic trees with rectangular edges as in Figure 3.2, this might not always be the user's wish—especially if he or she wants to use an unrooted layout. Therefore, the module offers the additional options of an *unrooted straight phylogram* and a *straight phylogram*. The main alteration to the rectangular style, is that the edge length (l) now does not only consist of the vertical distance (Δy) between two nodes, but also the horizontal portion (Δx). As the horizontal distance is, as mentioned before, determined by the *ideal sibling distance* and the diagonal length is determined by the evolutionary distance, the vertical length can be calculated with the Pythagorean theorem: $\Delta y = \sqrt{l^2 - (\Delta x)^2}$. This works fine as long as Δx is smaller than l—otherwise y is not real. In practice Δx may often be larger than l, if for example the scale of the picture is rather small, or if the nodes are very large and thus their centers need to be placed far from each other. Thus, to avoid error messages, if $\Delta x > l$ then $\Delta y = 0$.

The procedure of setting the positions is once more similar to the previously discussed ones. As the calculation of the y-positions depends on the x-positions, these need to be set before anything else is done; the y-positions are set subsequently.

Figure 3.3(b) shows an *unrooted straight phylogram*; the edge marked with an arrow delivers such a case where $\Delta x > l$ and thus this edge does not have the correct length. By increasing the *distance scaling factor*—which will be discussed in the following—this flaw can be eradicated, as depicted in Figure 3.3(d).

As the edge lengths depend on the distance matrix, the magnitude of the individual distances influence the total size of the tree, and hence, the picture. The Ti*k*Z scaling key must *not* be used to change the size, as it flaws the edge lengths, due to the fact that for example the font size is not influenced by the scaling key, neither are the nodes' *minimum sizes*. Thus, the tree is scaled non-uniformly and as the inner nodes are included in the edge length—as will be discussed a little further down—the ratio between the individual edges will be flawed. To solve this problem, there is another key, the *distance scaling factor*, which scales the phylogenetic distances and thus the edge lengths internally. Thus, if, for example, a distance of 1 should correspond to 2 cm on the paper, the factor must be set to `2cm`. The possibility of directly specifying the total tree size is sometimes also of great use. Thus, the implementation of a second scaling key should be considered. Such a scaling procedure, though, would be more complex than the implemented one: First the tree's coordinates would have to be set in order to determine the ratio between all the positions, and subsequently all the coordinates would have to be reset to the values that fit the required total tree size.

When using the layout option *straight phylogram* and the same input as in the previous examples, this results in the tree displayed in 3.3(c). The *distance scaling factor* would have to be three times larger in order to eliminate the faulty edge marked with an arrow. This result hints at the fact that the implementation of another, "smarter" graph drawing algorithm might be reasonable.

While every leaf's style can be directly manipulated by the user, this is not the case for the inner nodes, as they are not created by the user themself. Thus, to set the inner nodes' style the key *every phylogenetic inner node* must be used. When a new inner node is create by the algorithm, the style *phylogenetic inner node* is assigned to it, which is defined as follows:

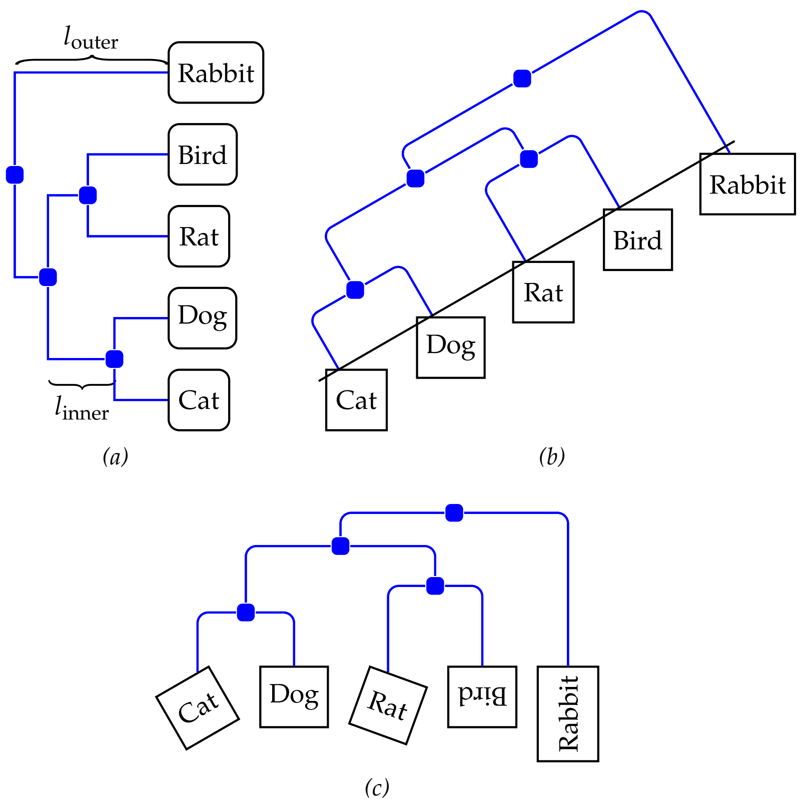


Figure 3.4: Defining the edge length in the picture and the effect of rotating on the determined edge length. For the distance matrix used for the computation see Figure C.2 in Appendix C. (The animal-name labels are purely fictional.) *(a):* Demonstration of what is included in inner edge lengths (l_{inner}) and outer edge lengths (l_{outer}), i.e. edges leading to leaves. *(b):* The effect of rotating the tree on the outer edge lengths. *(c):* The effect of rotating individual leaves on the outer edge lengths.

```
\pgfgdset{
    phylogenetic inner node/.style={
      /tikz/.cd, coordinate,
      every phylogenetic inner node/.try
   },
}
```

Hence, if the user does not specify anything else, inner nodes will be set to coordinates and thus be invisible. In case the tree is created without a phylogenetic algorithm, but instead by having the topology and edge lengths directly defined by the user, setting options to that key will have no effect: only nodes generated by the algorithm are internally marked as *phylogenetic inner nodes*. Nodes that are introduced by the user can be directly modified in any case.

In order to interpret the depiction of a tree correctly, it is important to define how exactly the evolutionary distance in terms of edge length is depicted. The graph drawing algorithms have been implemented in such a way, that the leaves themselves are *not* included in the edge length, as shown in Figure 3.4(a). This gives the user the opportunity of changing the leaves' appearance freely without interfering with the actual graph drawing computations. The inner edges' lengths, on the other hand, go from the center of a node (or coordinate) to another; see the edge labeled with l_{inner} in Figure 3.4(a). If the inner nodes are not overly large, as in the given examples, it is intuitive to take in the distance between two nodes as the distance between two layers. It might be worthwhile to consider adding the option of taking the inner nodes' sizes into account.

Another fact which should be noted is that the edge lengths are not correct if the tree is rotated in an angle not dividable by 90°. In Figure 3.4(b) a basis line has been drawn through the endpoint of the edge leading to the node *cat*; this line is orthogonal to all the edges connected to leaves. Thus, if the edge lengths were still correct, the line would go through all the edges' endpoints—which is not the case. The same goes for the rotation of individual nodes, as can be seen in Figure 3.4(c): the node "rabbit" is correctly placed, as it has been rotated by 90°, as well as "bird", which has been rotated by 180°. The edges leading to "rat" and "cat", however, are slightly too long. This, of course, is no mistake—Ti*k*Z aims at producing aesthetic pictures, and from that point of view the nodes in the two examples are perfectly aligned. As a rectangular placement of the tree and its leaves should in general be a more common aim than some angles in between, this problem can be bypassed.

4 CONCLUSION AND OUTLOOK

From the many different ways of calculating and visualizing phylogenetic trees, only a few are dealt with in this thesis. The module offers the possibility of having a tree constructed in a distance-based approach, either with UPGMA or with BME, and visualized by an algorithm designed by myself. In this section, these algorithms will be discussed and compared, leading to suggestions for further improvements.

Both UPGMA and BME depend on a distance matrix as their basic input. Even though the computation of the distance matrix is not part of the implemented module, its importance should not be forgotten: Without a valid distance matrix, no useful tree will be produced.

While UPGMA and BME may produce similar trees when the input matrix is very small, as is the case in this thesis' examples, their results deviate strongly from another if the number of taxa is significantly higher. As the appliance of such algorithms is usually relevant only if the number of taxa to-be-analyzed is *high*, the differences of the algorithms' working mechanisms are indeed relevant.

UPGMA is a greedy algorithm, and "mistakes" made during the tree-building are not corrected. BME, on the other hand, chooses the insertion point for every new leaf greedily as well, but the tree is constantly reconstructed and adjusted. BNNI optimizes BME's final result, by greedily choosing the best branch swap. Thus, especially for large trees, BME/BNNI are much more likely to find an optimal result than UPGMA [4, 7].

As mentioned before, UPGMA is based on the hypothesis that all taxa evolve at the same rate and thus all have the same distance to their root, i.e. their hypothetical common ancestor. This strongly limits the applicability of this approach [7]. BME, on the other hand, is actually a heuristic for finding the true *balanced minimum evolution tree* and has been shown to produce good results, i.e. with high topological accuracy, as shown by Desper and Gascuel [4]. It may be used as a good approximation to finding the true phylogeny of a set of taxa.

When it comes to comparing the algorithms' run times, BME is once more ahead of UPGMA. While the latter always has a time complexity of

$O(n^3)$, with n being the number of taxa[5], this is, in the worst case, also true for BME. But, depending on the tree diameter, i.e. the longest path in the tree, the time complexity may be a lot lower, and in practice it lies at $O(n^2 \cdot \log(n))$. BNNI, if run subsequent to BME, has a processing time of $O(p \cdot n \cdot \log(n))$, whereas p is the number of branch swaps, p is usually a lot smaller than n [4, 7].

It may be worthwhile to consider implementing an alternative way of inputting the distance matrix, as it is time-consuming for the author of a document to convert a large matrix into the Ti*k*Z distance key format. It could be allowed for to input the matrix in its actual form and have Lua convert it internally. This would increase the module's usability.

To further increase the module's applicability, one could implement possibilities to import phylogenetic trees, which were calculated by other programs, into Ti*k*Z. One potential way is the *Newick format*, a means of compactly describing trees [7], which is also used for phylogenetic algorithms. It would be useful to allow this syntax as input in Ti*k*Z and have Lua interpret and draw it, according to the user's specifications.

Let us turn once more to the graph drawing algorithms. The visualizing process is done by an algorithm designed by myself. Though it offers a few layout options, as presented in the previous section, it is rather limiting in some aspects, especially when it comes to displaying the edges in a non-rectangular way. Let us briefly take a look at a choice of other graph drawing algorithms; the following is mainly based on information taken from [7].

Next to the already implemented graph drawing algorithm for a rectangular phylogram, the *circular phylogram* is also often used. Here, every leaf gets an angle assigned to it, so that the leaves form a steady circle together. The root is placed in the middle, and the edges are drawn in such a way that only the radius represents the edge length. This circular form can be of advantage if the tree is very large. On the other hand, if the edge lengths are very short, there may be a lack of space to place the nodes nicely.

Unrooted trees, such as the ones produced by BME, can be depicted by *radial diagrams*. They are drawn similarly to circular phylograms, with the difference that the edges are drawn as straight lines only. An advantage of this graph drawing algorithm is that the leaves are placed in a well-distributed way. If the node sizes are taken into consideration, as they are in the implemented graph drawing algorithm, it might be difficult to assign the angles in such a regular way. Nevertheless, implementing a radial diagram algorithm would provide a good alternative for displaying unrooted trees.

All in all, the BME algorithm is superior to UPGMA, in terms of time complexity, biological relevance, and optimization. Which phylogenetic algorithm is applied to a given problem nevertheless depends on the specific aim, making the implementation of further algorithms reasonable.

Although there are other graph drawing algorithms which may be more suited for particular trees, the implemented algorithm does fulfill its pur-

[5]This stands for my and for the original implementation; Fionn Murtagh introduced a $O(n^2)$ solution [11].

pose. It is especially suited for rooted trees and thus for UPGMA, but has been adapted to also draw unrooted trees. In any case, a planar tree is always produced; the exemplary trees in this thesis can be seen as a demonstration.

The underlying structure of the implemented module will allow further graph drawing and phylogenetic algorithms to be incorporated easily in future: New layout and algorithm options can simply be added, and the respective new subclasses can then be called by *PhylogeneticTree*.

The possibility of producing phylogenetic trees directly in TEX is very convenient, if one needs to depict them for example in a thesis, such as this one. Being able to change the appearance of the trees at will, adjusting them to the complete document's style, is a valuable opportunity for anyone wishing to produce an appealing document.

5 REFERENCES

[1] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1999.

[2] Hans-Joachim Böckenhauer and Dirk Bongartz. *Algorithmic Aspects of Bioinformatics*. Springer-Verlag, 2007.

[3] Neil A. Campbell and Jane B. Reece. *Biologie*. Pearson Studium, 2009.

[4] Richard Desper and Olivier Gascuel. Fast and Accurate Phylogeny Reconstruction Algorithms Based on the Minimum-Evolution Princple. *Journal of Computational Biology*, 9(5):687–705, 2002.

[5] Richard Desper and Olivier Gascuel. Theoretical Foundation of the Balanced Minimum Evolution Method of Phylogenetic Inference and Its Relationship to Weighted Least-Squares Tree Fitting. *Molecular Biology and Evolution*, 21(3):587–598, 2004.

[6] Goossens, Mittelbach, and Samarin. *The LaTeX Companion*. Addison-Wesley Publishing Company, 1994.

[7] Daniel H. Huson, Regula Rupp, and Celine Scornavacca. *Phylogenetic Networks: Concepts, Algorithms and Applications*. Cambridge University Press, 2010.

[8] Roberto Ierusalimschy. *Programming in Lua*. Lua.org, second edition, 2006.

[9] Donald E. Knuth. *The TeXbook*. Addison-Wesley, 1986.

[10] LuaTeX development team. *LuaTeX Reference*, November 9, 2012. Available online at http://www.luatex.org/documentation.html.

[11] F. Murtagh. Complexities of Hierarchic Clustering Algorithms: State of the Art. *Computational Statistics Quarterly*, 1:101–113, 1984.

[12] Robert R. Sokal and Charles D. Michener. A Statistical Method for Evaluating Systematic Relationships. *University of Kansas Scientific Bulletin*, 38:1409–1438, 1958.

[13] Till Tantau. *The TikZ and PGF Packages. Manual for Version 2.10*, Oct. 25, 2010. Available online at http://sourceforge.net/projects/pgf/.

[14] Till Tantau. Graph Drawing in Ti*k*Z. In *Proceedings of Graph Drawing 2012*, Lecture Notes in Computer Science. Springer, 2012.

[15] Till Tantau. *The TikZ and PGF Packages. Manual for the Version currently under development*, Sept. 2012. Available online at http://sourceforge.net/projects/pgf/.

APPENDIX

A INSTALLING THE MODULE

Here is a quick description of how to install the module under Linux. The four class files

- pgf.gd.trees.BalancedMinimumEvolution2002.lua
- pgf.gd.trees.Maeusle2012.lua
- pgf.gd.trees.PhylogeneticTree.lua
- pgf.gd.trees.UPGMA1958.lua

should ideally be put in the correct pgf folder, i.e.:

pgf/generic/pgf/graphdrawing/lua/gd/trees.

Alternatively, they may also be in the same folder as the working TEX document.

The following must be put into the document's preamble:

```
\pgfgddeclarealgorithmkey
  {phylogenetic tree}
  {phylogenetic trees}
  {
    algorithm=pgf.gd.trees.PhylogeneticTree,
  }
\pgfgddeclareforwardedkeys{/graph drawing}{
  distances/.node parameter = table with node keys,
  distance scaling factor/.graph parameter = number,
  distance scaling factor/.parameter initial = 20,
  phylogenetic algorithm/.graph parameter = string,
  phylogenetic layout/.graph parameter = string,
}
\pgfgdset{
  inner node/.style={
    /tikz/.cd,
    coordinate,
    every phylogenetic inner node/.try},
}
```

B THE MODULE: OPTIONS AND COMMANDS

phylogenetic algorithm = ⟨string⟩

option	*description*
BME, Balanced Minimum Evolution, bme, balanced minimum evolution	Calls the BME algorithm and subsequently BNNI.
BME without BNNI	Calls BME without calling BNNI subsequently.
UPGMA, upgma	Calls the UPGMA algorithm.
none	No phylogenetic algorithm is called; the user needs to determine the tree structure and the desired edge lengths. This is the default value.

phylogenetic layout = ⟨string⟩

option	*description*
rectangular phylogram, rooted rectangular phylogram	
rooted straight phylogram, straight phylogram	Maeusle2012 is called and the x- and y-positions of the nodes are set according to the option.
unrooted rectangular phylogram	
unrooted straight phylogram	

distance scaling factor = ⟨number⟩

option	*description*
examples: 1 cm, 4 pt, ...	This factor scales the input distances. If the distance 1 should correspond to 4 cm on the paper, this value must thus be set to `4cm`.

distances = ⟨table with node keys⟩

option	*description*
examples: `{a = 1;}, {b = 4, c = 5;}, ...`	The phylogenetic distance of a node to other nodes, for example node a to nodes b and c: `a[distances = { b = 3, c = 4;} ];`

C DISTANCE MATRICES OF THE EXEMPLARY PHYLOGENETIC TREES

For the sake of completeness the following figures depict the distance matrices used for computation of the trees shown in Figures 2.7 and 3.4.

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*	*k*	*l*	*m*	*n*
a	0	4	9	9	9	9	9	5	4	8	7	7	4	3
b	4	0	9	9	9	9	9	3	4	8	7	7	4	2
c	9	9	0	2	7	7	7	2	7	3	6	4	6	5
d	9	9	2	0	7	7	7	5	6	5	8	3	7	4
e	9	9	7	7	0	3	5	4	4	7	6	9	8	8
f	9	9	7	7	3	0	7	6	4	5	7	5	5	5
g	9	9	7	7	5	7	0	4	8	7	6	5	4	5
h	5	3	2	5	4	6	4	0	4	5	6	7	4	6
i	4	4	7	6	4	4	8	4	0	6	4	4	4	6
j	8	8	3	5	7	5	7	5	6	0	6	4	6	4
k	7	7	6	8	6	7	6	6	4	6	0	6	4	6
l	7	7	4	3	9	5	5	7	4	4	6	0	6	5
m	4	4	6	7	8	5	4	4	4	6	4	6	0	5
n	3	2	5	4	8	5	5	6	6	4	6	5	5	0

Figure C.1: Distance matrix used as input for the BNNI example (Fig. 2.7).

	a	*b*	*c*	*d*	*e*
a	0	2	7	5	4
b	2	0	7	5	4
c	7	7	0	4	5
d	5	5	4	0	3
e	4	4	5	3	0

Figure C.2: Distance matrix used as input for the trees in Figure 3.4.