Eötvös Loránd University
Faculty of Science

Regina Krisztina Bíró

Constructing Phylogenetic Trees

BSc Thesis

Advisor: Kristóf Bérczi

Department of Operations Research

Budapest 2015

Acknowledgements

I would like to thank the following people for helping make this thesis possible:

Kristóf Bérczi, my supervisor, for advising me this topic and helping me throughout the process of making the thesis both mathematically and grammatically correct. His comments and valuable suggestions were extremely helpful for me.

My family for being understanding and supportive throughout my education.

My boyfriend and my friends for always being there for me.

Contents

Chapter 1

Introduction

According to biologists, there are about 5 to 100 million species of organisms living on Earth today. Evidence from morphological, biochemical, and gene sequence data suggests that all organisms on Earth are genetically related, and the genealogical relationships can be shown by a vast phylogenetic tree, the Tree of Life -a metaphor dating back over 100 years. At the leaves of this giant tree, are the species which are alive today. If we could trace their history back down the branches of the Tree of Life, we would encounter their ancestors, which lived thousands or millions years ago [15].

A **phylogenetic** or **evolutional tree** is a data structure shoving evolutionary relationships between biological species, or other entities. These relationships are called the **phylogeny** of the species and are based upon differences or similarities in their genetic or physical characteristics [11].

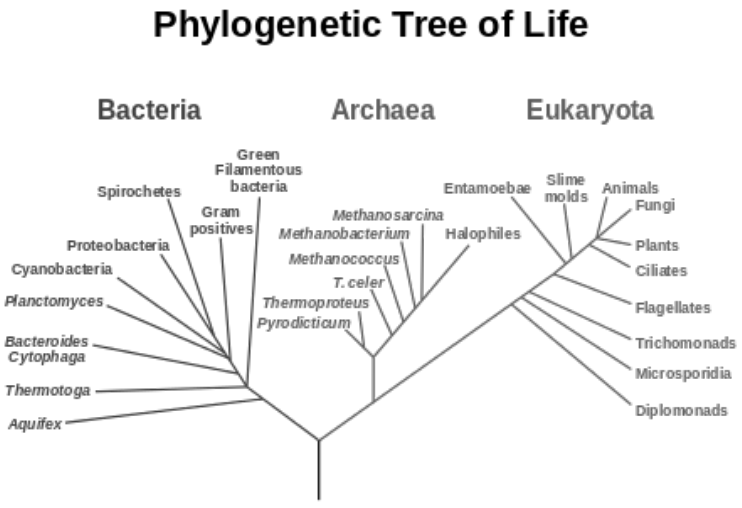

Figure 1.1: An example of a phylogenetic tree [11]

Phylogeny could be considered the 'proof' of evolution, and evolution is the central unifying principle of biology. It has several major applications within biological sciences,

such as biological classification. It can also be applied to human understanding of life, of biochemistry, and of evolution. Many biotechnological applications also benefit from studies of phylogeny, and its applications in the field of medicine may directly impact patients' lives. Phylogeny and its methods, such as cladistics, are important fields of science [19].

The main objective of the thesis is to present a detailed survey about the methods that can be applied to construct a phylogenetic tree from given input sequences. We will also discuss the effectiveness of these methods in practical use. The first and most critical step in the procedure is to find an adaptable sequence alignment, which gives the data used for the tree construction. In Chapter 2, the most common sequence alignment methods will be described via dynamic programming. Chapter 3 focuses on different methods available in the literature for constructing a phylogenetic tree. The most important and widely used methods will be described in details, such as Neighbor Joining and Maximum Parsimony and the related problems to it (Perfect phylogeny, the Steiner tree problem in phylogeny). A linear programming approach for solving the maximum parsimony problem will also be discussed.

Chapter 2

Sequence Alignments

In bioinformatics, **sequence alignment** is a method for arranging the sequences of DNA, RNA, or protein to identify regions of similarity that may be a consequence of functional, structural, or evolutionary relationships between them [27]. These sequences, for example DNA, contain information of living cells. When a cell duplicates, its DNA molecules double, and both daughter cells contain a copy of the original cell's DNA. However, this replication is not perfect, because the stored information can change due to random mutations. These mutations are the cause of the vast diversity of the species living on Earth [33].

2.1 Pairwise sequence alignment

Given two sequences as an input, we can ask how many mutations are needed to transform one into the other. There are several types of mutations, and one can occur more frequently than the other. We assign weights to the different the types of mutations: frequent mutations get lower weights, rare mutations get greater weights. The weight of a series of mutations is defined as the sum of the weights of the individual mutations. The task is to find the minimum weight series of mutations, which transform one sequence into another. An important question is that how can we quickly find such a minimum weight series. A greedy algorithm finds all the possible series of mutations, and then chooses the minimum weight, but it is too slow, since the possible number of series of mutations grows exponentially. Therefore, dynamic programming is used to get a global solution [33].

2.1.1 Dynamic programming

Dynamic programming is often used to solve optimization problems, such as finding the shortest path between two points. Instead of solving the original problem, it defines and solves a set of new, smaller subproblems, then combines the solutions of the generated subproblems to reach an overall solution.

Without solving a subproblem multiple times, it reduces the number of computations: it solves each subproblem only once, and stores the results. The next time when a solution of a subproblem is needed, it can be simply looked up. This method is especially useful when the number of subproblems grows exponentially [3, 30].

With given sequences $X = (x_1, \ldots, x_n)$ and $Y = (y_1, \ldots, y_m)$, the task is to compute the difference or similarity between them. For every character, **cost functions** of insertions (δ) and transpositions ($\alpha(x_i, y_j)$) are given. We assume that X is transformed by the above operations to Y, and we are looking for the minimum cost transformation. $S(i)$ denotes the first i characters of a sequence S, and $opt(i, j)$ denotes the cost of transforming $X(i)$ into $Y(j)$. Starting with $opt(0, 0) := 0$, $opt(i, 0) := 0$ and $opt(0, j) := 0$, we get the value of $opt(i, j)$ from the results of the previous transforming optimums, with the transposition of x_i and y_j. We add the cost of this transformation to the value of $opt(i-1, j-1)$:

$$\textbf{opt(i,j)} = min\{\alpha(x_i, y_j) + opt(i-1, j-1), \delta + opt(i-1, j), \delta + opt(i, j-1)\},$$
$$i \geq 1,\ j \geq 1.$$

The running time of the algorithm is $O(nm)$, since $opt(i, j)$ is computed in constant time [30].

For sequence alignment, we define a finite set of symbols Σ, and a set Σ^* of finite sequences over Σ. When comparing two DNA or protein sequences, the following transformations can be applied:

- Insertion of a character x before position i, denoted by ${}_ix \leftarrow -$.
- Deletion of a character x at position i, denoted by ${}_i- \leftarrow x$.
- Substitution of character x with character y at position i, denoted by ${}_iy \leftrightarrow x$.

The symbol $\circ$ denotes the concatenations of mutations, and τ denotes a set of concatenations of the above mutations. $T(A) = B$ denotes that sequence A can be transformed with $T \in \tau$ into B. We define a weight function $w : \tau \to \mathbb{R}^+ \cup \{0\}$. The **transformation distance** between A and B is the minimum weight of transformations turning A into B:

$$\delta(A, B) = min\{w(T)|,\ T \in \tau,\ T(A) = B\}$$

If we assume that w satisfies the following

$$w(b \leftarrow a) = w(a \leftarrow b)$$
$$w(a \leftarrow a) = 0$$
$$w(b \leftarrow a) + w(c \leftarrow b) \geq w(c \leftarrow a)$$

for any $a, b, c \in \Sigma \cup \{-\}$, then δ is a metric on Σ^*. Hence, it suffices to consider only transformations that change each position of a sequence at most once. These series of

transformations are visualized by **sequence alignments**. A pairwise sequence alignment is displayed below - by convention, the ancestor is the sequence on top, and the descendant is the sequence at the bottom.

A	U	C	G	U	C	-	G	A	A
A	-	G	G	U	A	A	C	A	-

For example, this alignment shows that there were substitutions at positions three, six and eight, deletions at positions two and ten, and there was an insertion at the seventh position. We call a pair of characters at a certain position an **aligned pair**. The weight of the series of transformations is the sum of the weights of aligned pairs. Thus, it suffices to find a minimum weight alignment of A and B, instead of finding the minimum weight transformation between them [33].

Definition 1. An **alignment** of A and B is a pair of sequences of equal length over $\Sigma \cup \{-\}$. The non-gap characters of the first sequence give back A, and the non-gap characters of the second sequence give back B. Furthermore, there is no position in which both sequences contain the gap symbol. We call an alignment **optimal**, if it's weight is minimal.

Let $\alpha^*(A_i, B_j)$ denote the set of optimal alignments of A_i and B_j, and $w(\alpha^*(A_i, B_j))$ denote the weights of alignments in $\alpha^*(A_i, B_j)$. Like we described previously, the method for finding an optimal alignment is that we calculate $w(\alpha^*(A_i, B_j))$ from the previous results: $w(\alpha^*(A_{i-1}, B_j))$, $w(\alpha^*(A_i, B_{j-1}))$ and $w(\alpha^*(A_{i-1}, B_{j-1}))$. This calculation can be done in constant time. It works, because if we delete the last aligned pair of an optimal alignment of A_i and B_j, we get the optimal alignment of A_{i-1} and B_j or A_i and B_{j-1} or A_{i-1} and B_{j-1}, depending on what the deleted aligned pair depicted: a deletion, an insertion, substitution or match. Hence

$$\begin{aligned} w(\alpha^*(A_i, B_j)) = min\{&w(\alpha^*(A_{i-1}, B_j)) + w(- \leftarrow a_i), \\ &w(\alpha^*(A_i, B_{j-1})) + w(b_j \leftarrow -), \\ &w(\alpha^*(A_{i-1}, B_{j-1})) + w(b_j \leftarrow a_i)\}. \end{aligned}$$

We calculate the weights of optimal aligments in a D$\in \mathbb{R}^{(n+1)\times(m+1)}$ matrix, where n and m denote the lengths of the aligned sequences. This matrix is the so-called **dynamic programming table**, where the $d_{i,j}$ element ($d_{i,j}$ is the entry at the intersection of column i and row j) contains $w(\alpha^*(A_i, B_j))$ and the rows and columns are indexed from 0 to n and m, respectively. The initial conditions for column 0 and row 0 are the following:

$$d_{0,0} = 0$$
$$d_{i,0} = \sum_{k=1}^{i} w(- \leftarrow a_k)$$
$$d_{0,j} = \sum_{l=1}^{j} w(b_l \leftarrow -)$$

The table can be filled using recursion

$$\begin{aligned} d_{i,j} = min\{&d_{i-1,j} + w(- \leftarrow a_i), \\ &d_{i,j-1} + w(b_j \leftarrow -), \\ &d_{i-1,j-1} + w(b_j \leftarrow a_j)\} \end{aligned}$$

The time requirement of the fill-in is $O(mn)$. After the dynamic programming table is filled, we can found the optimal alignments with the following method, called trace-back: we go from the right bottom corner to the left top corner choosing cell(s) which give the optimal value of the current cell. Stepping up from position $d_{i,j}$ means a deletion, stepping to the left means an insertion, and the diagonal steps can mean either a substitution or a match, depending on whether or not a_i=b_j. We can represent these steps with directed edges, thus we get a directed graph whose vertices are the cells of the dynamic programming table. The set of optimal alignments can be represented in polynomial time and space, despite the fact that the number of them might grow exponentially with the length of the sequences.

There are several versions of dynamic programming algorithms applied for pairwise sequence alignments, since the input sequences can vary: different weight functions, like gap penalties (weighting deletions and insertions in different ways) can be introduced [33].

2.2 Multiple sequence alignment

The multiple sequence alignment problem was first introduced in the early 70's, and by today it became one of the most important problems in bioinformatics. Multiple alignments are common in searching databases and deducing evolutionary relationships. It is much more relevant than pairwise sequence alignment, since there are biologically important patterns that cannot be revealed by the comparison of two strings, but they become clear when many related strings are simultaneously compared. Moreover, in some cases it is possible to demonstrate continuous changes between two strings that by themselves does not show much similarity [26]. With the use of multiple alignments, we can also find

the retained parts of a family of sequences, the positions which describe the functional properties of the sequence family [33].

Definition 2. A **multiple alignment** of $k \geq 2$ strings $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$ is a natural generalization of the alignment for two strings. Gaps are inserted into (or at either end of) each of the k strings so that the resulting strings have the same length, defined to be l. Then the strings are arrayed in k rows of l columns each, so that each character and gap of each string is in a unique column [26].

The dynamic programming approach for the optimal pairwise alignment problem is the basis of the dynamic programming approach for the optimal multiple alignment. To align k sequences, we have to fill a k-dimensional dynamic programming table (a k-dimensional hypercube), which contains an entry for each combination of the prefixes, and stores their optimal alignment. Thus the memory requirement for k sequences of length n is $\Theta(n^k)$ [33].

Even if we assume that Evolution is reversible, it is not obvious to find a good scoring for multiple sequence alignment. We can describe the evolutionary relationships of the sequences by a rooted binary tree.

Definition 3. A **rooted binary tree** is a tree in which the degree of all but one internal nodes is 3, and one internal node has degree 2. The root of the tree is the node with degree 2. The degree 1 nodes are called **leaves**.

In this representation the leaves of the rooted binary tree stand for the modern species and the root of the tree represents their common ancestor. We should define the score of an alignment column depending on the evolutionary relationship among the species. However, this kind of scoring is a chicken-egg problem: we want to obtain the evolutionary relationship between the sequences we align, but it is impossible to define a score by their evolutionary relationships without knowing it. There are several less elegant but widespread methods for solving this problem, the most common one is the **sum-of-pairs scoring (SP)**. In an alignment column, it calculates a score for each pairs of characters, and sums them [33].

The other problem with the multiple sequence alignment is that it has a running time that grows exponentially with the number of sequences.

Theorem 4. *The problem of existence of a multiple alignment with Sum-of-pairs score (SP-score) is NP-complete.*

Proof. We prove that the following version of the multiple alignment problem with SP-score is NP-complete. Let $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$ denote a set of sequences and let c denote a positive integer. The question is: is there a multiple alignment for $\mathcal{S}$ with value at most c? Now we reduce the shortest common supersequence problem to multiple alignment with SP-score.

Definition 5. Given a string S over an alphabet Σ, a **supersequence** S' of S is a string $S' = w_0x_1w_1 \dots x_kw_k$ over Σ such that $S = x_1x_2 \dots x_k$ and each $w_i \in \Sigma^*$.
A **common supersequence** of a set of strings $\mathcal{S} = \{S_1, S_2, \dots S_k\}$ is a string $\mathcal{S}$ over Σ such that S is a supersequence of each S_i.

Definition 6. For a finite set of sequences $\mathcal{S}$ over alphabet Σ and a positive integer m, the **shortest common supersequence problem** is to decide whether there is a sequence S with $|S| \leq m$ such that each $t = t_1t_2 \dots t_r \in \mathcal{S}$ is a subsequence of S (i.e. $S = s_0t_1s_1 \dots t_rs_r$, for some $s_0, s_1, \dots, s_r$).

The proof of the NP-completeness of the shortest common supersequence problem can be found in [36]. The problem remains NP-complete even if $|\Sigma|$ =2 [32].

Now we can continue the proof. Obviously, multiple alignment is in NP. The reduction from the shortest common supersequence problem is the following. Let $\mathcal{S}$ denote a set of sequences over alphabet $\Sigma = \{0, 1\}$, and let m denote a positive integer. We construct a collection of sets $X = \{X_{i,j} | i, j \geq 0, i + j = m\}$, where $X_{i,j} = \mathcal{S} \cup \{a^i, b^j\}$, where a and b are two newly introduced letters. We may assume that each sequence in $\mathcal{S}$ has length at most m. Table 2.1 shows the scores we construct for the differences of the characters.

$\mathcal{S}$	0	1	a	b	Δ
0	2	2	1	2	1
1	2	2	2	1	1
a	1	2	0	2	1
b	2	1	2	0	1
Δ	1	1	1	1	0

Table 2.1: The score scheme

This score scheme satisfies triangle inequality. We define the positive integer c required for the modified multiple alignment problem as $c = (k-1)||\mathcal{S}|| + (2k+1)m$, where $||\mathcal{S}||$ denotes the total length of all sequences in $\mathcal{S}$.

For proving the NP-hardness of the multiple sequence alignment, it suffices to show that $\mathcal{S}$ has a supersequence of length m if and only if some $X_{i,j}$ has an alignment with value at most c.

($\Rightarrow$) Suppose that we have an alignment $\mathcal{A}$ of the $k+2$ sequences in $X_{i,j}$, with value at most c for some (i, j). Now we consider the induced alignment of the k sequences in $\mathcal{S}$. This alignment has always score $(k-1)||\mathcal{S}||$. Consequently, the total contribution of pairwise alignments in $\mathcal{A}$ involving the two sequences a^i and b^j is at most $(2k+1)m$. Therefore, every 0 must be aligned with an a and every 1 must be aligned with a b in $\mathcal{A}$. A supersequence S for $\mathcal{S}$ now can be obtained by assigning 0 to the columns containing a's and 1 to the other columns. The length of supersequence S is $i + j = m$.

($\Leftarrow$) Let S be a supersequence for $\mathcal{S}$ with length m. Let i denote the number of 0's and let j denote the 1's in S. Consider the set $X_{i,j}$. For each sequence $t \in \mathcal{S}$, an alignment of t and S exists such that each 0 in X_i (or 1 in X_j) matches a 0 (or respectively a 1) in S. Some of the characters 0 and 1 can correspond to spaces. In order to obtain the desired multiple alignment, we align each t in $\mathcal{S}$ with S, then we align the a's in the sequence a^i with the 0's in the sequence S, and respectively do the same for the b's in b^j and the 1's in S. In this alignment the letters in the columns are either $0, a$ and Δ or $1, b$ and Δ. If we remove sequence S from the alignment, the value will be exactly c.

Thus, by checking the value of an optimal alignment of $X_{i,j}$ where $i + j = m$, we can decide in polynomial time if there is a supersequence S of length m in X [41]. □

Chapter 3

Constructing a Phylogenetic Tree

There are several methods for tree construction, but most of them can be classified as either **phenetic** (or in other words, distance-based) or **cladistic** (character-based) [38]. Figure 3.1 sums up the most widespread methods and their relations.

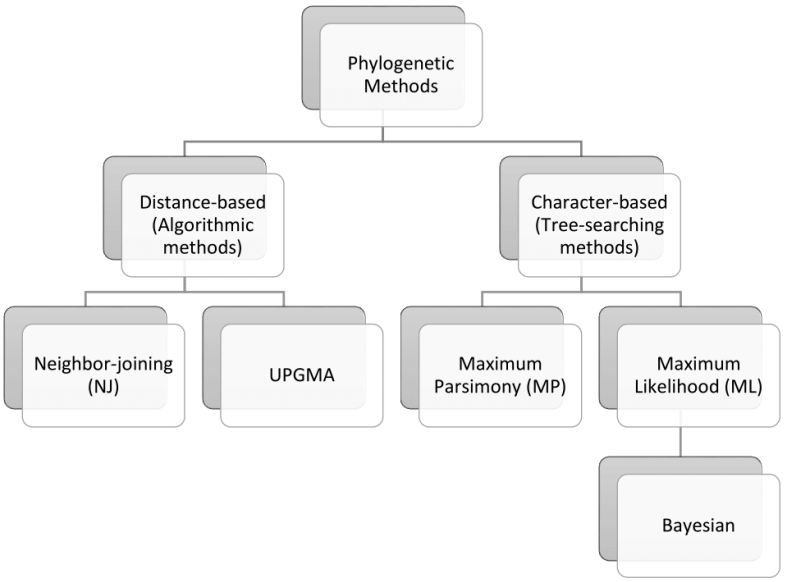

Figure 3.1: An overview of the major approaches of tree-construction [38]

Before describing them in detail, Table 3.1 gives a comparison between the main approaches based on their strengths and weaknesses.

3.1 Phenetic Methods

Character-based methods use algorithms that integrate evolutionary models, like (amino acid) substitution, and compute a distance matrix (as described in Chapter 2). From this

Method	**Strength**	**Weakness**
Distance-based methods	• Uses a molecular evolution model • Provides branch length	• Provides only a single tree • Different trees may result depending on the order of the sequence entry • Branch presented as distances as opposed to discrete characters • Cannot infer ancestral states • Cannot identify individual characters that are informative or problematic
Character-based methods		
Maximum parsimony	• Minimizes the number of steps and thus the number of additional hypothesis • Identifies numerous parsimonious trees from which a consensus is calculated • Provides diagnoses for each clade and branch lengths in terms of the number of changes on each branch of a tree • Can infer ancestral states • Can identify individual characters that are informative or problematic	• Relatively slow compared with NJ — particularly when dealing with large data sets • Lack an explicit model of evolution • As more divergent sequences are analyzed, the degree of homoplasty increases — in this case the true evolutionary tree becomes less likely to be the one with the least number of changes and parsimony fails • Highly unequal mutation rates associated with rapidly evolving lineages may lead to "long branch attraction"
Maximum likelihood	• Strong statistical foundations • Allows comparison of different trees, parameters and models • Uses all of the sequence data	• Significantly slower than alternative methods and computationally demanding

Table 3.1: A comparison between the main approaches [38]

matrix, the phylogenetic tree is calculated with progressive clustering.

The applied evolutionary model specifies the substitutions that occurred in the sequences since they began to differ from the last common ancestor. Based on this information, the distances in the matrix refer to the number of differences (where substitutions might have happened) between the sequences. From this numerical data, a tree is constructed by the following rule: the most closely related sequences are positioned on the tree in a way that they are more distant from the other sequences [38].

3.1.1 Unweighted pair group method with arithmetic mean (UPGMA)

The method UPGMA is a hierarchical clustering method. At each step, it makes a higher-level cluster from the closest two clusters. Figure 3.2 shows the clustered structure of a UPGMA tree. The approach defines the distance between two clusters A and B to be the mean distance between elements of each cluster, which means the average of all the distances between all objects $x \in A$ and $y \in B$:

$$d(A, B) = \frac{1}{|A| + |B|} \sum_{x \in A} \sum_{y \in B} d(x, y)$$

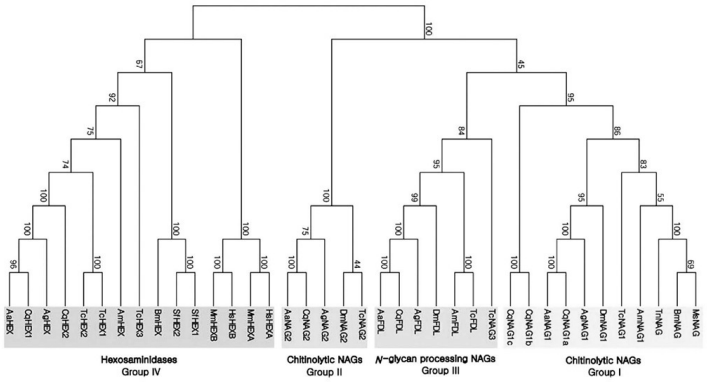

Figure 3.2: An example of a phylogenetic tree constructed with UPGMA [1]

The method uses the „Molecular clock hypothesis" as an evolutionary model, which means that it assumes a constant rate of evolution. In phylogenetics, it is not a widely used approach since it assumes certain relationships between the sequences, without these relationships being tested for the actual data. For this reason it is only used to produce guide-trees in the process of other sophisticated phylogenetic constructional approaches [13]. It was shown that the optimal time for constructing the UPGMA tree is $O(n^2)$ [35].

3.1.2 The Neighbor Joining method

Neighbor Joining (NJ) was introduced in 1987 by Saitou and Nei. This approach has become the most popular method for building phylogenetic trees, and since it was published, the original paper has been cited over 13000 times [23]. On Figure 3.3, we can see the genetic distance graph of the 18 human races based on 23 types of genetic information. It was made with neighbor-joining method in 2002 by Saitou Naruya, japanese professor at the Japanese National Institute for Genetics [9].

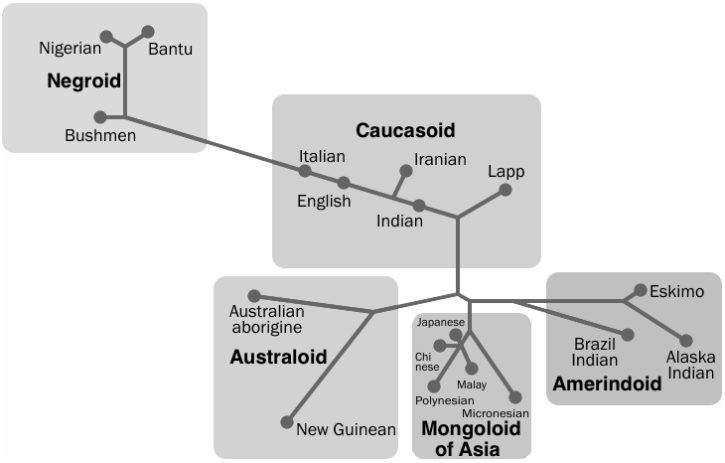

Figure 3.3: An example of a phylogenetic tree constructed with NJ [9]

NJ builds a tree from the distance matrix: it contains the pairwise evolutionary distances between the elements of a set of sequences. When starting, we have a star-like tree (as seen in Figure 3.4) which is yet unresolved. The algorithm iterates the following steps:

1. First, the algorithm calculates a new matrix Q as follows. Based on the distance matrix with n sequences, compute

$$Q(i,j) = (n-2)d(i,j) - \sum_{k=1}^{n} d(i,k) - \sum_{k=1}^{n} d(j,k),$$

where $d(i,j)$ is the distance between the ith and jth sequences.

2. It finds a pair i and j $(i \neq j)$ of sequences for which $Q(i,j)$ has the minimum value. Now we create a new node attached to the central node, and attach the ith and jth nodes to it. In Figure 3.4 f and g denote these nodes, as being attached to the new node u.

3. We calculate the distance of the sequences attached to the new node from the other sequences using formula

$$\delta(f,u) = \frac{1}{2}d(f,g) + \frac{1}{2(n-2)}[\sum_{k=1}^{n} d(f,k) - \sum_{k=1}^{n} d(g,k)],$$
$$\delta(g,u) = \delta(f,g) - \delta(f,u),$$

where f and g denote the paired nodes joined to u, $\delta(f,u)$ and $\delta(g,u)$ denote the branch length between its two nodes, and these lengths will not be affected by the later steps of the construction.

4. Now we calculate the distances of the new node u from the other sequences (which are not f and g). For each sequence k that was not involved in the previous step, let

$$d(u,k) = \frac{1}{2}[d(f,k) + d(g,k) - d(f,g)].$$

5. Now we iterate these steps using the newly generated node u instead of f and g, with the recalculated distances [9].

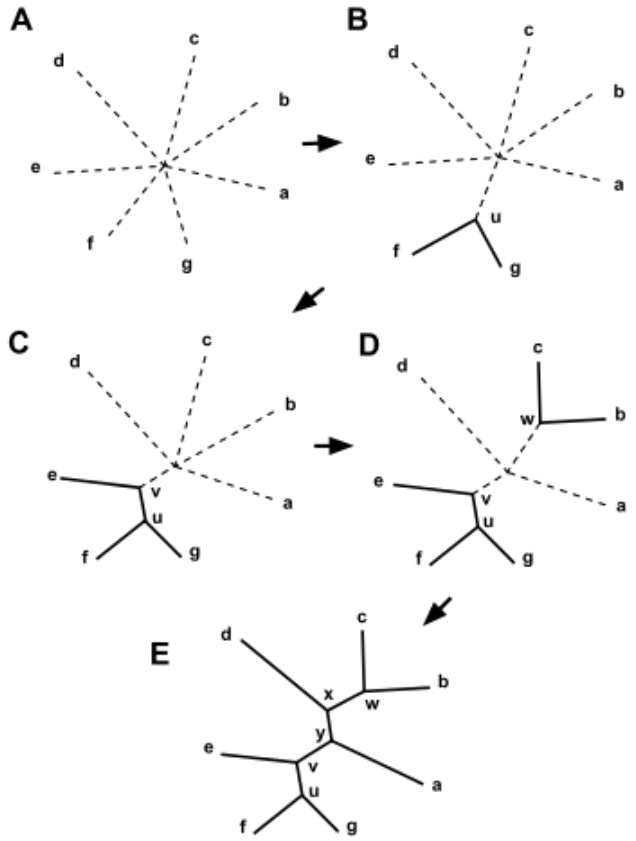

Figure 3.4: The process of tree building with NJ [9]

For a set of n sequences, the algorithm iterates $(n-3)$ times. First, it constructs an $n \times n$ size matrix Q, than a $(n-1) \times (n-1)$ size one, etc. This implementation leads to time complexity $O(n^3)$, but there exist implementations that are a lot faster in average [9].

The main advantage of NJ is that it is fast and provides a polynomial-time algorithm. This makes it practical compared to the character-based methods, when a larger set of data (hundreds or thousands of sequences) is being analyzed. In practice, the input distance matrix is rarely correct, but if it is, which means that the matrix is 'nearly additive', and in addition when the pairwise distances in the matrix differ from the true measurements by less than half of the shortest branch length in the tree, the neighbor joining method always gives the topologically correct tree. Even if the distance matrix is not correct, NJ often finds the correct tree topology anyway. This correctness means that by using dissimilar evolutionary models and sufficiently long input data, there is a high probability that NJ constructs the real tree. Compared to the other distance-based method, UPGMA, it has

an advantage that it does not use the 'molecular clock hypothesis' as it does not assume that all changes happen at the same rate [9].

Despite these advantages, distance-based methods in general has been superseded by other approaches which do not rely on distance data, yet provide high accuracy. There are other significant limitations for NJ, as it computes only one tree, while character-based methods generate several optimal or nearly optimal trees and choose the best one. Furthermore, NJ has the undesirable feature to compute different trees depending on the order of the input sequences, and also assigns negative values for some branch lengths in the tree [9, 38].

3.2 Cladistic Methods

Based on an evolutionary model and a sequence alignment, cladistic or character-based methods find the tree which has the biggest probability to be optimal for a specific input set of sequences. Since they are searching for an optimal tree, they are often called tree-searching methods. The most common cladistic approaches are Maximum Parsimony (MP), Maximum Likelihood (ML) and the group of Bayesian methods, which are variants of the ML method [38]. From these approaches, only the MP method will be described in details altogether with the problems related to it. We only give a brief overview on ML. The description of Bayesian methods falls outside the scope of the thesis, since we are focusing on computational approaches.

In these approaches, the input is a set of attributes called **characters** that the investigated objects may possess. If these input characters are well-chosen, then the information made of the distribution of the characteristics among the objects could deduce partial evolutionary history - and it also forms a phylogenetic tree. Before starting to describe the approaches, an important question is that where do character data come from?

When investigating biological objects (for example, species), traditionally morphological features as characters are used. These morphological characters could be general attributions like possessing a backbone or very specific features that only specialists understand [26]. As these characters often describe an attribute or feature that objects either do or do not possess, character-based problems are often **binary-character problems**.

Definition 7. Let M be a $0-1$ matrix of size $n \times m$ representing n objects in terms of m characters or traits that describe the objects. Each character takes on one of two possible states 0 or 1, and cell (p, i) of M has value 1 if and only if object p has character i [26].

Definition 8. Given an n by m binary-character matrix M for n objects, a **phylogenetic tree for M** (see Figure 3.5) is a rooted tree T with exactly n leaves that obeys the following properties.

1. Each of the n objects labels exactly one leaf of T.
2. Each of the m characters labels exactly one edge of T.
3. For any object p, the characters that label the edges along the unique path from the root to leaf p specify all of the characters of p that is 1 [26].

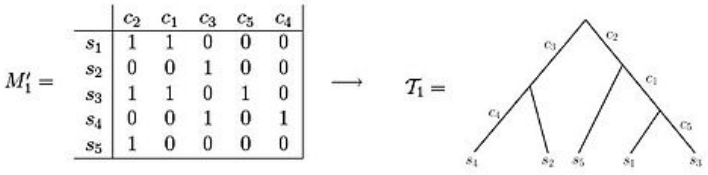

Figure 3.5: A binary matrix and a phylogenetic tree built from it [10]

This interpretation of a phylogenetic tree for a matrix gives an estimation of the evolutionary history of the objects, and is based on the following assumptions

1. The root of the tree represents a specific ancestral object which has none of the present characters. Thus, in this ancestral object, the state of each character is zero.

2. Each of the characters changes from state 0 to state 1 exactly once, and never changes from 1 to 0.

The key feature, without which we would not consider the problem interesting, is that each character labels exactly one edge of the tree. This edge represents an evolutionary change, when one character (represented by this edge) converts from state zero into state one. This means, by the second assumption, that any objects below that edge definitely have that character. It is a very difficult task to find a set of characters that satisfies these assumptions [26].

Characters can be also based on sequence data, like DNS or protein sequences of the different species. For instance, examining whether or not a sequence contains a specific substring makes a good binary character. However, characters based on morphological features can be ambiguous as under some conditions similar morphological features can develop independently or a feature could be lost and regained over time. For example, according to the current accepted opinion in biology, wings have evolved independently several times. Therefore, substrings in DNA sequences make a better approach for character data in character-based reconstructions than morphological traits do.

In evolutionary studies, an increasingly important character type is the nucleotide type in a certain position of a DNA sequence. The examined DNA sequences are generally first multiply aligned. After the process, each column of the alignment specifies a single character, which can be one of the states A, T, C and G. For any given species p, the state of character i is the nucleotide at the position (p, i) in the multiple alignment. These characters are not binary, but in some cases DNA sequences are reduced to binaries by grouping A with G (these are called purines) and C with T (the pyrimidines). Most of the problems in Phylogeny are binary character problems, like the Perfect Phylogeny Problem in Section 3.2.2.1, which is a special case of the general maximum-parsimony problem [26].

3.2.1 Maximum Likelihood

Maximum likelihood estimation is a statistical method which is used to estimate the parameters of a statistical model. It is a frequently used method in a wide range of topics, like several statistical models (for example, linear models and discrete choice models), communication systems, data modeling in nuclear physics, and of course, computational phylogenetics.

In phylogenetics, the maximum likelihood method uses statistical techniques and assigns probabilities for a group of possible phylogenetic trees. It is especially accurate for building molecular phylogenies. According to several computer studies, ML has the ability to find a correct tree in a relatively short time, and works fine for distantly related data sets. Another strengths of the method is that it can compare different trees with different evolutionary models within a statistical framework. However, the drawback is that even with small data sets, it is impossible for the method to find the optimal tree for sure and since it requires to search all the combinations of tree topology and branch length, it is a computationally expensive method if used for more than a few sequences. There are several heuristics to fasten the approach, like branch-and-bound or the pruning algorithm, which is a variant of dynamic programming. It was not shown to be NP-complete to search for optimal tree topologies defined by likelihood, but finding an optimal tree is still challenging as branch-and bound search is not yet practical for the accustomed representation of phylogenetic trees [2, 5, 24].

3.2.2 Maximum Parsimony

Maximum parsimony (MP) is a character-based method that builds a phylogenetic tree by minimizing the total tree length. It searches for the minimum number of evolutionary steps required to explain a given set of data. These steps are for instance substitutions between DNA sequences. The approach searches for all of the possible tree topologies from the given input data, and chooses the optimal (minimal) tree. The optimum is usually called the **most parsimonious tree** [7]. Searching for the optimal tree can be computationally hard, because the number of possible trees rapidly grows with the number of input sequences. For example, the number of rooted trees in case of n input sequences is

$$N_r = \frac{(2n-3)!}{2e^{(n-2)}(n-2)!},$$

while the number of unrooted trees is

$$N_u = \frac{(2n-5)!}{2e^{(n-3)}(n-3)!}.$$

This rapid growth makes it impossible to apply the method on vast data sets (Table 3.2.2 shows that even for small input sets the number of trees to be analyzed grows very fast). When applied, maximum parsimony describes a non-parametric statistical method for possible relations. However, it is not consistent, since the probability of producing the evolutionary correct tree is not very high under certain conditions [6, 7].

Number of sequences	Number of rooted trees	Number of unrooted trees
2	1	1
3	1	3
4	3	15
5	15	105
6	105	945
7	954	10396

Table 3.2: The number of possible trees increases rapidly with the number of inputs [6]

Maximum Likelihood can be an alternate when a problem cannot be solved with Maximum Parsimony. There is a connection between ML and MP: if we add to the set of sequences some constant sequences (in which all characters are the same) and apply the maximum likelihood method, then it will choose the most parsimonious tree [37].

In the next sections we first introduce the **perfect phylogeny problem** (Section 3.2.2.1), which is a special case of maximum parsimony, then we provide a clean definition of the maximum parsimony problem by introducing a class of problems called **Steiner tree** problems.

3.2.2.1 The Perfect Phylogeny Problem

Definition 9. (Perfect phylogeny problem)
Given an $n \times m$, $0-1$ matrix M, determine whether there is a phylogenetic tree for M, and if so, build one [26].

The perfect phylogeny problem can be solved with an $O(nm)$-time algorithm, where it takes one time unit to compare two elements. In order to be correct, first we will reorder the columns of M. Since M is a $0-1$ -matrix, we consider its columns as binary numbers. We sort these M numbers into a decreasing order (thus, the largest number will be the first column). M' denotes the reordered version of M. It is easy to see that m has a phylogenetic tree if and only if M' does. We will name each character by the column containing it in M'. For example, character j will be to the right side in M' of every character i if and only if $i < j$ [26].

Definition 10. For any column k of M', let O_k denote the set of objects having state 1 in column k (i.e. the set of objects that possess character k).

If O_k strictly contains O_j, column k must be placed left from column j. Moreover, if a column has a duplicate, they are placed together in a block following one another in M'. Now we state the main theorem which is the core for solving the perfect phylogeny problem.

Theorem 11. *M' (or M) has a phylogenetic tree if and only if for every pair of columns i and j, either O_i and O_j are disjoint or one contains the other.*

Proof. We suppose that T is a phylogenetic tree for M', and consider the characters i and j. The edge of T on which character i is changed from state 0 to state 1 is denoted by e_i. All the objects that possess character i are placed under the edge e_i. For i and j one of the following four cases must hold:

1. $e_i = e_j$,
2. e_i is on the path from the root to e_j,
3. e_j is on the path from the root to e_i,
4. the paths of the two individual edges diverge before reaching either e_i or e_j.

In case 1, the objects which possess character i and j are identical, so $O_i = O_j$. In case 2, the objects which possess character j must also possess character i, so $O_j \subset O_i$. Case 3 is symmetrical to case 2, so $O_i \subset O_j$. In case 4, $O_i \cap O_j = \varnothing$. In all the cases either O_i and O_j are disjoint or one contains another.

The other direction will be proved by construction. We consider the objects p and q, and let k be the largest character in M' that both objects p and q possess. We state that if p possesses a character $i < k$, then q must contain character i. Namely, since p contains both i and j, O_i and O_k intersect. Thus O_i must contain O_k and q also possesses character i. Since p and q are arbitrary choices, if q possesses a character $i < k$ then p must also possess i. Summarizing, $M'(p, i) = M'(q, i)$ for every character $i \leq k$, and $M'(p, j) = M'(q, j)$ for $j > k$ if and only if $M'(p, j) = M'(q, j) = 0$.

In order to finish the proof, we will use **keyword trees**.

Definition 12. The keyword tree for a set P of patterns is a rooted directed tree K satisfying three conditions:

1. each edge is labeled with exactly one character,
2. any two edges leaving the same node have distinct labels,
3. every pattern P_i in P maps to some node v of K such that the characters on the path from the root to v exactly spell out P_i, and every leaf of K is mapped to some pattern in P.

An object p gets labeled by a string, which consists of the characters possessed by p, in the order they appear in M'. In order to be sure that no resulting string is a prefix of an other string, we also add an end-of-string character, let it be Ω, which was not in the original alphabet. We already showed that the strings for objects p and q must be identical

until some character k, and after k have no characters in common. Thus, the keyword tree for the n strings constructed from n objects in M' specifies a perfect phylogeny for M'. We can obtain the perfect phylogeny from the keyword tree by removing the Ω symbols from the tree edges [26]. □

Now we present an $O(mn)$-time algorithm for the perfect phylogeny problem:

1. Consider each column of M a binary number. With **radix sort**, sort these number into decreasing order, so the largest number will be in the first column. Call the resulting matrix M' and name each character based on its column position in M'.
2. For each row p of M', construct a string that consists of the characters (in increasing order) that p possess.
3. Build the keyword tree T from the n strings constructed in the previous step.
4. Test if T is a perfect phylogeny for M.

This algorithm can be implemented in $O(mn)$ time if we use radix sort with pointers [26].

3.2.2.2 Application and generalization of perfect phylogeny

Several tree-building approaches have been introduced for finding a phylogenetic tree. However, these different methods often give different trees with different software packages. When building a phylogenetic tree, a method relies on the comparison of a single position in the aligned sequences. Hence, very often the resulting tree will differ in some detail, depending on which position was used. It is an important task to determine whether two (or more) different phylogenetic trees describe a corresponding evolutionary history. Moreover, if they do, how to combine the trees into a single phylogenetic tree which incorporates all the known history.

When establishing a theory of evolutionary history, it is generally accepted that several trees (from different positions in sequences) have to be built and shown to be correspondent in order to consider the evolutionary theory reliable. This topic is called **tree compatibility**, and is an application of perfect phylogeny [26].

Definition 13. A phylogenetic tree T' is a **refinement** of T if T can be obtained by a series of contractions of edges of T'.

If a tree T' refines T, it means that T' is consistent with all the evolutionary history stored in T, while T' contains additional information which is not contained in T. Let T_1 and T_2 denote two phylogenetic trees built from a set of n objects. Assume that both T_1 and T_2 are binary trees, and no node except the root can have exactly one child.

Definition 14. T_1 and T_2 are **compatible** if there exists a phylogenetic tree T_3 refining both T_1 and T_2.

Definition 15. Tree compatibility problem: With given trees T_1 and T_2, determine whether they are compatible, and if so, give a refinement tree T_3.

Let M_1 be a binary matrix with n rows (for each object) and one column for each internal node in T_1. The (i, j) value of M_1 is 1 if and only if the leaf of object i is at a position below node j. Thus, column j of M_1 consists of the objects which can be found in the j-rooted subtree of T_1. For T_2, matrix M_2 can be constructed similarly. Matrix M_3 is formed by the union of the columns of M_1 ad M_2.

Theorem 16. *T_1 and T_2 are compatible if and only if there is a phylogenetic tree for M_3, and a phylogenetic tree T_3 for M_3 is a refinement for both T_1 and T_2.*

Theorem 16 reduces the tree compatibility problem to the perfect phylogeny problem, and implies an $O(n^2)$ algorithm for solving the compatibility problem by applying the algorithm described for perfect phylogeny [26].

In perfect phylogeny, as we previously described, each character can possess two states. In the **generalized phylogeny problem**, however, a character is allowed to take on more than two states. Thus, a perfect phylogeny for M will be a directed tree T, in which each object labels exactly one leaf of T as seen before, but in this case we label the edges with **character-state transitions**. More precisely, we label an edge with an ordered tripe (c, x, y) meaning that character c changes from state x to state y along the labeled edge. Just like in the binary case, each character has a starting state at the root of the tree. Each leaf on the path P between the root and a leaf p describes the character states of an object p. It is required that the ending states on each edge of the path to p correctly describes the character states of object p. For every state y of a character c there is at most one edge where the previous state of c changes to state y. This also means that there can be at most one edge labeled with a triple starting with c and ending with y.

In the binary case, the triples labeling an edge for character c are all $(c, 0, 1)$ [26].

Definition 17. The generalized phylogeny problem. Given a character matrix M where each character may take on up to r states, determine if there is a perfect phylogeny for M, and if so, construct one.

The generalized perfect phylogeny problem was first proposed in 1974 and remained an open issue for almost 20 years [20]. Later results showed polynomial-time solution in term of n and m (the number and length of sequences), if r (the number of character states) is fixed to 3 or 4 [21, 29].

However, if r is a variable, the perfect phylogeny problem is NP-complete [18]. In later studies it was shown that if r is any fixed value, the perfect phylogeny problem can be solved in polynomial time in terms of n and m, however, r appears as an exponent in the worst-case time bound [28].

3.2.2.3 The Steiner Problem in Phylogeny

Definition 18. Let $G = (N, E)$ denote an undirected graph with $|N|$ nodes and $|E|$ edges, with a weight $w(i, j)$ on each edge (i, j) in E. Let $X \subseteq N$ be a given subset of nodes. A **Steiner tree** ST for X is a connected subtree of G which contains all the vertices of X, and may contain other nodes from $N - X$ as well. The **weight of a Steiner tree** ST is the sum of the weights on the edges contained by ST, and is denoted by $W(ST)$. Given a graph G and a subset of nodes X, the **weighted Steiner tree problem** is to find a Steiner tree of minimum weight. In the case of all edges having weight 1, the problem is called **unweighted Steiner tree problem**.

Definition 19. A d-dimensional **hypercube** is an undirected graph with 2^d nodes, where they are labeled with integers between 0 and $2^d - 1$. Two nodes are connected if and only if their labels as binary numbers differ in precisely one bit.

Definition 20. The **weighted Steiner problem on hypercubes** is the weighted Steiner tree problem where G is a hypercube. The unweighted case is when all edges have weight 1.

As previously mentioned, maximum parsimony is a problem of evolution history reconstruction with the minimum number of mutations. A more precise definition can be given using the concept of Steiner trees.

Definition 21. For a set X of input sequences of length d and binary character states, the **maximum-parsimony problem** is the unweighted Steiner tree problem on a d-dimensional hypercube. Each sequence of X can be described by a d-length binary vector and thus it describes one node in the hypercube [26].

A d-dimensional hypercube has 2^d nodes, so 2^d possible objects could be described by a d-length binary vector. Two sequences that differ in exactly one character will be denoted by two adjacent nodes in the hypercube. Thus, a Steiner tree connecting a given set of X with l edges exists if and only if there is a suitable phylogenetic tree that describes l character-state mutations.

For a set of input sequences, each containing d binary characters, when each of the characters are nontrivial (the case when a character is contained by some, but not all of

the sequences), the constructed maximum-parsimony tree must have at least d mutations. Consequently, we can view the perfect phylogeny problem as an equivalent problem for finding a maximum-parsimony, whose cost equals to exactly d, which is the lower bound for the costs. In the generalized case, we define the lower bound based on the fact that if a character has r states, then there will be $r-1$ places in the maximum parsimony tree where the state-changes occur

As a result, from the viewpoint of maximum-parsimony and the Steiner tree problems, perfect phylogeny answers the question whether the optimal Steiner tree has a greater cost than the lower bound. For the Steiner tree-problem there is no efficient solution either on unweighted graphs or hypercubes in particular. When r is fixed, then approximations for the generalized perfect phylogeny problem show (see Section 3.2.2.2) that this special problem for Steiner trees can be solved in polynomial time [26].

The unweighted Steiner tree problem on hypercubes was shown to be NP-complete in 1982 [22]. Normally, the complexity of an unweighted problem implies that the weighted problem is at least as complex, since the unweighted case can be solved via the weighted case by setting all weights to 1. In the case of the Steiner tree problem, the NP-completeness of the unweighted problem does not imply that the weighted problem is NP-complete (which in fact, is NP-hard), as was shown in [22].

Despite of the NP-hardness of the weighted Steiner tree problem on hypercubes, an efficient algorithm with an approximation ratio less than two exists, as the weighted Steiner tree problem on an arbitrary graph can be approximated within a bound of 11/6 [42]. Prior to this result, the problem was approximated by the fact that the minimum spanning tree can be used to obtain a Steiner tree if its weight is less than twice the weight of the optimal Steiner tree [31].

In case of a hypercube, the method is the following.

1. Compute the distance $d(i,j)$ in the hypercube between each pair of input sequences i and j from X.
2. Let K_X denote a complete, undirected graph in which a node represents an object in X, and the weight of an edge (i,j) is $d(i,j)$ (the distances).
3. Compute a minimum spanning tree T of K_X (here, each edge of T corresponds to a path in the hypercube).
4. Transform T by expanding its edges to their original paths in the hypercube.
5. Form a graph G' from the paths found in Step 4, and search for a spanning tree of G'.

The result is a Steiner tree of the set X which has total weight less than twice the weight of an optimal Steiner tree [26]. In case of binary characters, it is unnecessary to use hypercubes

as this computation can be done more efficiently. The approximations shown in [22] and [17] can also be adapted for solving the maximum-parsimony problem, without explicitly using hypercubes [26].

3.2.2.4 Finding the most parsimonious tree via Linear Programming

In this section, an integer linear programming approach will be introduced (based on a paper by Sridhar, Lam, Blelloch, Ravi and Schwartz [40]) for finding the maximum parsimony (or the most parsimonious phylogenetic tree) from a set of binary data. There have been a lot of heuristics (the description of them falls out the scope of this thesis) introduced for the maximum parsimony problem, and although they are efficient, they cannot guarantee the optimal tree. The following linear programming method is efficient even on larger datasets.

Maximum parsimony and its variants are NP-hard, consequently, there has been no efficient solutions found. It can be very time-consuming, since the examined data-sets tend to be extremely large both in population sizes and numbers of variations examined. As described in the previous section, the maximum parsimony problem with binary-element sequences is usually modeled by binary Steiner trees, which is an NP-hard problem. Only some special cases can be solved, which we called the perfect phylogeny problem in Section 3.2.2.1. However, real data-sets usually do not correspond to the perfect phylogeny assumption. Some progress have been made by introducing the concept of **near-perfect phylogenies**, which means that they differ by a fixed value from the optimum. These approaches are efficient for moderate input size, but not for larger data-sets. Thus integer linear programming methods (ILP) got a bigger attention, since they provide optimal solutions. Although these methods do not have time-bound guarantees, they tend to have better running times than other methods also granting optimality.

The input is a binary matrix H, as its columns correspond to the Single Nucleotide Polymorphism (SNP) which means a character can change from state 0 to state 1.

Definition 22. The length of a phylogenetic tree T is the sum of Hamming-distances between each adjacent nodes. The problem of constructing **the most parsimonious phylogenetic tree** is to find a minimum-length phylogenetic tree T^*.

Definition 23. The **length** of a phylogenetic tree $T(V, E)$ is $length(T) = \sum[w_i : i \in D(u, v), uv \in E]$, where $D(u, v)$ is the set of indices in which u and v differ.

Definition 24. A phylogenetic tree T for input I with m-length sequences is **q-near perfect** (or q-imperfect) if $length(T) = m + q$.

In Section 3.2.2.2, it was pointed out that the maximum parsimony problem is equivalent to the minimum Steiner tree problem in a G graph on vertices V_T. G is an $m-$ *hypercube* defined on $V = \{0,1\}^m$ nodes, where $V_T \subseteq V$ corresponds to the sequences and each two nodes are adjacent if and only if they differ in exactly one bit. Although the Steiner-problem is NP-complete [22], the case when the phylogenetic tree is q-near perfect can be solved in polynomial time in terms of n and m when $q = O(log(poly(n,m)))$ [39]. However, when q is large, these approaches do not perform well.

The first step of the integer linear programming method is preprocessing - this step reduces the size of the input data without affecting the final result. When solving the Steiner tree problem, there are exponentially many possible subsets that can be chosen as the Steiner nodes (V_T) and this has an influence on the complexity. The first task is to eliminate those vertices that cannot be accepted in an optimal tree.

The method which will be used to eliminate unnecessary vertices works on m-cubes. For an input graph H and a column c of H a **split** $c(0)|c(1)$ defined by column c is a partition of the sequences into two sets, where $c(0)$ is a set of sequences which have 0 in the cth position and $c(1)$ is the set of sequences which have 1 in column c. The sets $c(0)$ and $c(1)$ are called **blocks of c** and the graph constructed from the blocks is called **Buneman-graph** $\mathcal{B}(H)$ (see Figure 3.6) which contains the structural features of the optimal phylogeny. Here a generalization of the Buneman-graph will be constructed as the following

1. Each node of the Buneman-graph is an m-tuple of blocks: $c_1(i_1), c_2(i_2), \ldots, c_m(i_m)$, where $i_j = 0$ or $i_j = 1$ for all $1 \leq j \leq m$.

2. There is one block for each column, such that any two blocks have a nonempty intersection: $c_j(i_j) \cap c_k(i_k) \neq \varnothing$ for all $1 \leq j, k \leq m$.

3. Two vertices in $\mathcal{B}(H)$ are adjacent if and only if they differ in exactly one block.

The following theorem shows why Buneman-graphs are important.

Theorem 25. *For an input matrix H let T_H^* denote the optimal phylogenetic tree on H and let $\mathcal{B}(H)$ denote the Buneman-graph on H. If H has binary values, then all of the optimal phylogenetic trees T_H^* are subgraphs of $\mathcal{B}(H)$.*

By Theorem 25, the problem can be reduced to constructing a Buneman-graph on an input matrix H and then solving the problem on $\mathcal{B}(H)$. Now we show that a Buneman-graph can be constructed efficiently.

Theorem 26. *The Buneman-graph $\mathcal{B}(H)$ is a connected graph if in the input matrix H all columns contains both states 0 and 1, and all pairs of columns are distinct.*

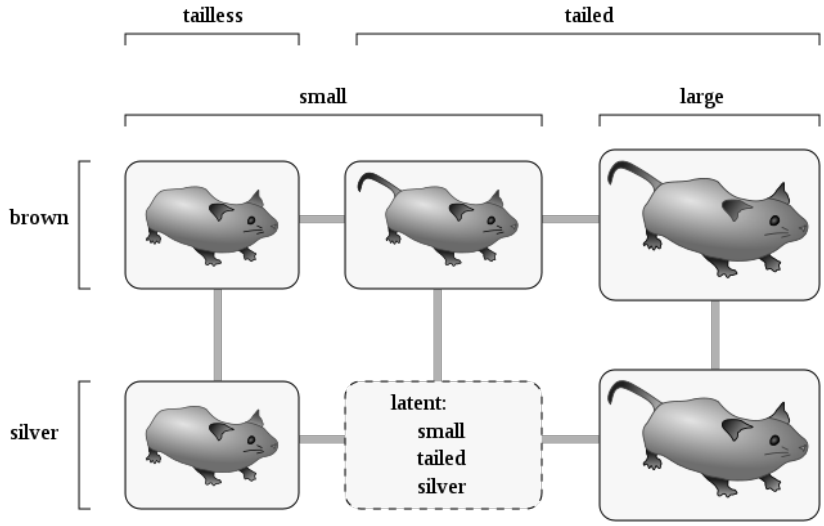

Figure 3.6: An example of a Buneman graph for five types of mouse [8]

---

function findBuneman(V_T)

1. $\lambda \leftarrow V_T$, $v \in \lambda$
2. BunemanNeighbor(λ, v)

function BunemanNeighbor(λ, v)

for all $j \in \{1, \ldots, m\}$

(a) let $v' \leftarrow v$, $v'_j \leftarrow c_j(1 - i_j)$

(b) If v' is Buneman and $v' \notin \lambda$ then

i. $\lambda \leftarrow \lambda \cup v'$

ii. BunemanNeighbor(λ, v')

Figure 3.7: Construction of the Buneman-graph [40]

To construct the graph $\mathcal{B}(H)$, let $i_1, i_2, \ldots, i_m$ be the first sequence in H. It follows that $v = [c_1(i_1), c_2(i_2), \ldots, c_m(i_m)]$ is a vertex of $\mathcal{B}(H)$. There are several ways to construct the Buneman-graph $\mathcal{B}(H)$. The procedure described in Figure 3.7 begins with a vertex set V_T from $\mathcal{B}(H)$ consistent with H. The algorithm iteratively selects a vertex v and enumerates all its neighbors. The algorithm checks if a neighbor satisfies the conditions of the Buneman-graph, and if so, adds it to λ, then does a recursion.

Lemma 1. *The procedure shown in Figure 3.7 finds the Buneman-graph $\mathcal{B}(H)$ for the given input H in time $O(km)$, where k is the number of vertices in $\mathcal{B}(H)$.*

Proof. The algorithm starts with vertex $v \in \mathcal{B}(H)$, and computes $\mathcal{B}(H)$ in a depth-first-search order. Theorem 26 implies that all the vertices of $\mathcal{B}(H)$ will be examined. It visits all the m possible neighbors of v in the m-hypercube, which takes $O(m)$ time. For each

vertex $v \in \mathcal{B}(H)$, the algorithm calls the function $BunemanNeighbor$ only once. Thus, if $\mathcal{B}(H)$ has k vertices, the time required for going over $\mathcal{B}(H)$ is $O(km)$. □

In addition to the reduction of the set of vertices of the Steiner tree problem, Theorem 25 implies that the problem can be divided into smaller, independent subproblems.

Definition 27. A pair of columns i and j **conflict** if the matrix H restricted to these two columns contains all the following pairs of binaries: $(0,0), (0,1), (1,0)$, and $(1,1)$. Equivalently, columns i and j conflict if the projection of H onto dimensions i and j contains all the four corners of the square.

The structure of conflicts in an input I plays a significant role in providing information for the construction of the optimal phylogenetic tree for I. In [25] it was shown that a perfect phylogeny exists if and only if no pair of the columns conflict. In order to represent the conflicts of H, a **conflict graph** $\mathcal{G}$ will be constructed. In $\mathcal{G}$ the vertices correspond to the columns of H, and two vertices i and j are adjacent if and only if the columns i and j conflict in H.

Theorem 28. *Let χ denote the set of nontrivial connected components of the conflict graph $\mathcal{G}$, and let V_{isol} denote the set of isolated vertices of $\mathcal{G}$. An optimal Steiner tree on H is the union of optimal Steiner trees on the separate components of $\mathcal{G}$, and $length(T_{H*}) = |V_{isol}| + \sum[length(T^*_{C(H)} : C \in \chi]$, where $C(H)$ denotes the matrix H restricted to the set of columns C.*

The decomposition steps are the following.

1. First construct the conflict graph $\mathcal{G}$ for input H, and search for the set of connected components in $\mathcal{G}$.

2. Ignore the columns which correspond to the isolated vertices V_{isol} as they each add exactly one edge to the final phylogeny.

3. The columns corresponding to each connected component C can be solved independently on order to attain the perfect phylogeny.

Thus, the problem is now decomposed into matrices H_i consisting of a nontrivial connected component.

The next step is to transform matrix H_i (from now, it will be denoted as H) in order to reduce its size. Rows can be removed from H until all rows are distinct, as this does not change the phylogenetic tree. Also, those trivial columns can be removed which do not contain both stages 0 and 1 as they will not affect the size and the topology of the resulting phylogenetic tree.

Weights w_i will be assigned to each column i. Set $w_i = 1$ for each i. The following steps will be performed iteratively: identify columns i and j tat are identical (up to relabeling with 0 or 1), set $w_i := w_i + w_j$ and remove column j from H. In the resulting matrix H, each pair of rows are distinct, moreover each pair of columns are distinct (even after relabeling 0 and 1), all of the columns contain both states 0 and 1, and every column i has weight $w_i \geq 1$. Hence the new input will be a matrix H and a vector w that contains the weights of the columns of H.

Lemma 2. *The length of the optimal phylogenetic tree on the pre-processed input is the same as on the original input.*

In the ILP formulation, a more general problem will be considered. We will use directed weighted graphs for finding the minimal Steiner tree, by representing an undirected graph with a directed graph by replacing one edge with two directed edges. The input for the directed minimum Steiner tree problem consists of a directed graph with a set of terminals T and a specified root vertex $r \in T$. The minimum Steiner tree will be the minimum cost subgraph that contains a directed path from r to all other terminal vertices in T.

Let $x^S \in \mathbb{R}^E$ denote a vector (for a subgraph S) whose elements are edge variables x_e^S and have value 1 if the subgraph S contains edge e, otherwise 0. A subset of vertices $U \subset V$ is **proper** if it is nonempty and does not contain all the vertices of V. For a subset $U \subset V$, $\delta^+(U)$ denotes the set of edges (u, v), with $u \in U$ and $v \notin U$. For a subset of edges $F \subseteq E$, $x(F) = \sum_{e \in F} x_e$. There is also a weight function on the edges denoted by $w \in \mathbb{R}^E$.

A graph cut-based ILP formulation was used in [16] for the problem of finding a minimum directed r-rooted Steiner-tree:

$$min \sum_{u,v} w_{u,v} x_{u,w} \tag{1}$$

$$\text{subject to } x(\delta^+(U)) \geq 1, \forall \text{ proper } U \subset V \text{ with } r \in U, \text{ where } T \cap U \neq \emptyset \tag{2}$$

$$x_{u,v} \in \{0, 1\}, \text{ for all } (u, v) \in E. \tag{3}$$

The second constraint implies that r has got a directed path towards all terminal vertices T. Since the underlying graph here is a Buneman graph, any of the vertices can be chosen to be the root r. The Buneman graph may have an exponential number of vertices, and edges proportional to the size of the input matrix H. Thus, the running time for solving this integer programming problem in worst case is doubly-exponential in m.

Now an alternative solution will be described using multicommodity flows. In this case, one unit of flow is sent from r to every terminal. Every terminal vertex except for r will be a sink for exactly one unit of flow, and all the Steiner vertices have perfect flow conservation. Introduce the following variables denote for every edge $(u, v) \in E$:

1. $f^t_{u,v}$ are real valued, and represent the amount of flow on edge (u, v) directed towards a terminal t,

2. $s_{u,v}$ are binary valued and denote the presence or absence of edge (u, v).

The program is the following

$$min \sum_{u,v} w_{u,v} s_{u,v} \tag{4}$$

$$\text{subject to } \sum_v f^t_{u,v} = \sum_v f^t_{v,u} \text{ for all } u \notin T \tag{5}$$

$$\sum_v f^t_{v,t} = 1, \sum_v f^t_{t,v} = 0, \sum_v f^t_{r,v} = 1 \text{ for all } t \in T \tag{6}$$

$$0 \leq f^t_{u,v} \leq s_{u,v} \text{ for all } t \in T \tag{7}$$

$$s_{u,v} \in \{0, 1\} \text{ for all } e \in E \tag{8}$$

Constraint (5) implies that the flow is conserved on the Steiner vertices. Constraint (6) is the inflow and outflow constraint for terminals in T. Constraint (7) implies that there is positive flow on an edge only if the edge is present in the graph. According to the max-flow-min-cut theorem, the solution projected onto the variables s satisfy the constraints described by (2).

Theorem 29. *All integer variables in the above linear program are binary and the solution to the ILP gives the most parsimonious tree.*

The authors of [40] has applied this ILP formulation on several versions of datasets, and implemented the algorithm in C++. In each case, the ILP method was able to find the optimal tree after preprocessing. This success shows that this method fills an important practical need for fast methods which can find the optimal tree in reasonable time, even for larger data sets.

Chapter 4

Conclusion

Phylogeny is one of the most powerful tools in medicine research. It helps understanding the spread of infectious diseases, or analyzing the evolution of several viruses which is very important when designing vaccinates. It is also very helpful in the protection and diversity conservation of endangered ecosystems or species, since phylogeny can aid in selecting the prior species for monitoring and protecting [14].

The future prospects of the studies in phylogeny is merging phylogenetics with genomics (a genome is the total genetic content in a set of chromosomes [34]) and is called **phylogenomics**. Phylogenetics focuses on the presence of a single phylogenetic marker, an attribution or a characteristic as described in Chapter 3. On the contrary, phylogenomic approaches aim to compare a whole gene content, and to investigate the presence or absence of a set of gene families and/or gene order [38].

The approaches described in this thesis are based on the Darwinian-Mendelian model, which assumes only vertical gene-transfers (changes occur from the parent to offspring), but recent studies have shown that mutational events, like LGT (Lateral Gene Transfer, also called Horizontal Gene Transfer, as it occurs horizontally, showed on Figure 4.1) strongly influence evolution, and thus should not be ignored.

LGT is also the primary reason for bacterial antibiotic resistance and a significant driver of genetic variation. The traditional phylogenetic approaches have difficulties when it comes to distinguishing genes that are similar because of LGT from the genes that are similar because they share a common ancestor. Although the standard phylogenetic methods are effective in computing phylogenetic trees among closely related species, they have drawbacks when comparing more distantly related organisms, as a consequence of lateral gene transfer and the variing rates of evolution for different genes. A solution for this problem is to compare large number of genes or entire genomes among many species, as laterally transferred sequences behave differently from expected, and these anomalies stand out from the pattern of evolution indicated by the majority of the data. As a consequence, the results given by a more robust phylogenomic method will not be as influenced by these

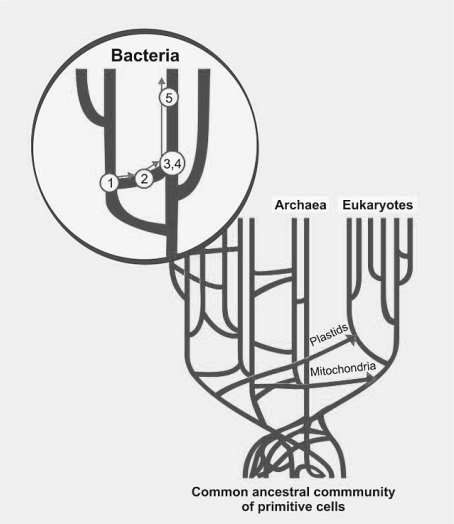

Figure 4.1: Vertical and horizontal gene transfers [4]

mutations as a simple phylogeny.

These studies show that instead of the concept of 'Tree of Life', phylogenetic analysis must consider the network or forest indicated by the variety of vertical and horizontal gene transfers in order to 'see the wood from the trees' [4, 12, 38].

Bibliography

[1] Chitin metabolism in insects part 4. http://what-when-how.com/insect-molecular-biology-and-biochemistry/chitin-metabolism-in-insects-part-4/. [Online; Accessed 2015-03-06]. 12

[2] Computational phylogenetics — Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Computational_phylogenetics. [Online; Accessed 2015-03-07]. 18

[3] Dynamic programming — Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Dynamic_programming. [Online; Accessed 2014-11-30]. 4

[4] Horizontal gene transfers — Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Horizontal_gene_transfer. [Online; Accessed 2015-04-19]. 32

[5] Maximum likelihood — Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Maximum_likelihood. [Online; Accessed 2015-03-07]. 18

[6] Maximum parsimony analysis. http://www.icp.ucl.ac.be/~opperd/private/parsimony.html. [Online; Accessed 2015-03-13]. 18, 19

[7] Maximum parsimony (phylogenetics) — Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Maximum_parsimony_(phylogenetics). [Online; Accessed 2015-03-13]. 18

[8] Median graph — Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Median_graph. [Online; Accessed 2015-04-25]. 27

[9] Neighbor joining — Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Neighbor_joining. [Online; Accessed 2015-02-26]. 12, 13, 14, 15

[10] Perfect phylogeny — Wikipedia, the free encyclopedia. 16

[11] Phylogenetic tree — Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Phylogenetic_tree. [Online; Accessed 2014-11-20]. 1

[12] Phylogenomics — Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Phylogenomics. [Online; Accessed 2015-04-19]. 32

[13] Upgma — Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/UPGMA. [Online; Accessed 2015-02-25]. 12

[14] Uses of phylogenies: Medicine. http://www.zo.utexas.edu/courses/evolution/applied.pdf. [Online; Accessed 2015-04-17]. 31

[15] What is phylogeny? http://tolweb.org/tree/learn/concepts/whatisphylogeny.html. [Online; Accessed 2014-11-20]. 1

[16] J. E. Beasley. An algorithm for the steiner problem in graphs. *Networks*, 14(1):147–159, 1984. 29

[17] P. Berman and V. Ramaiyer. Improved approximations for the steiner tree problem. In *Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms*, pages 325–334. Society for Industrial and Applied Mathematics, 1992. 25

[18] H. L. Bodlaender, M. R. Fellows, and T. J. Warnow. *Two strikes against perfect phylogeny*. Springer, 1992. 23

[19] R. Broyles. What is phylogeny used for? http://www.brighthubeducation.com/science-homework-help/8880-what-is-phylogeny-used-for/. [Online; Accessed 2014-11-20]. 2

[20] P. Buneman. A characterisation of rigid circuit graphs. *Discrete mathematics*, 9(3):205–212, 1974. 22

[21] A. Dress and M. Steel. Convex tree realizations of partitions. *Applied Mathematics Letters*, 5(3):3–6, 1992. 22

[22] L. R. Foulds and R. L. Graham. The steiner problem in phylogeny is np-complete. *Advances in Applied Mathematics*, 3(1):43–49, 1982. 24, 25, 26

[23] O. Gascuel and M. Steel. Neighbor-joining revealed. *Molecular biology and evolution*, 23(11):1997–2000, 2006. 12

[24] S. Guindon and O. Gascuel. A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood. *Systematic biology*, 52(5):696–704, 2003. 18

[25] D. Gusfield. Efficient algorithms for inferring evolutionary trees. *Networks*, 21(1):19–28, 1991. 28

[26] D. Gusfield. *Algorithms on strings, trees and sequences: computer science and computational biology*. Cambridge university press, 1997. 6, 7, 16, 17, 19, 21, 22, 23, 24, 25

[27] Z. Hongwei. Molecular phylogenetics. http://abc.cbi.pku.edu.cn/talk/phylogeny-zhu-hw.pdf. [Online; Accessed 2014-11-14]. 3

[28] S. Kannan and T. Warnow. A fast algorithm for the computation and enumeration of perfect phylogenies. *SIAM Journal on Computing*, 26(6):1749–1763, 1997. 23

[29] S. K. Kannan and T. J. Warnow. Inferring evolutionary history from dna sequences. *SIAM Journal on Computing*, 23(4):713–737, 1994. 22

[30] A. Kis-Benedek. Dinamikus programozás a gráfelméletben, 2010. Bsc Thesis. 4

[31] L. Kou, G. Markowsky, and L. Berman. A fast algorithm for steiner trees. *Acta informatica*, 15(2):141–145, 1981. 24

[32] M. Middendorf. More on the complexity of common superstring and supersequence problems. *Theoretical Computer Science*, 125(2):205–228, 1994. 8

[33] I. Miklós. Introduction to algorithms in bioinformatics, 2010. Electronic notes. 3, 5, 6, 7

[34] W. Morris. American heritage dictionary of the english language. 1969. 31

[35] F. Murtagh. Complexities of hierarchic clustering algorithms: State of the art. *Computational Statistics Quarterly*, 1(2):101–113, 1984. 12

[36] K.-J. Räihä and E. Ukkonen. The shortest common supersequence problem over binary alphabet is np-complete. *Theoretical Computer Science*, 16(2):187–198, 1981. 8

[37] S. Roch. A short proof that phylogenetic tree reconstruction by maximum likelihood is hard. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 3(1):92, 2006. 19

[38] R. D. Sleator. Phylogenetics. *Archives of microbiology*, 193(4):235–239, 2011. 10, 11, 15, 16, 31, 32

[39] S. Sridhar, K. Dhamdhere, G. E. Blelloch, E. Halperin, R. Ravi, and R. Schwartz. Simple reconstruction of binary near-perfect phylogenetic trees. In *Computational Science–ICCS 2006*, pages 799–806. Springer, 2006. 26

[40] S. Sridhar, F. Lam, G. E. Blelloch, R. Ravi, and R. Schwartz. Efficiently finding the most parsimonious phylogenetic tree via linear programming. In *Bioinformatics Research and Applications*, pages 37–48. Springer, 2007. 25, 27, 30

[41] L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *Journal of computational biology*, 1(4):337–348, 1994. 9

[42] A. Z. Zelikovsky. An 11/6-approximation algorithm for the network steiner problem. *Algorithmica*, 9(5):463–470, 1993. 24