Algorithms for Phylogenetic Reconstructions

Lecture Notes and Exercises

Lectures by Sven Rahmann
Exercises by Constantin Bannert

Previous versions by:
Martin Vingron, Jens Stoye, Hannes Luz, Sebastian Böcker

Winter 2005/2006

Preface

These lecture notes are the result of a collaborative effort of many people. They result from a series of lectures given by Martin Vingron (MPI/FU Berlin) and Jens Stoye (Bielefeld University) and a practical course by Hannes Luz (MPI, Berlin), and have subsequently been extended by Sebastian Böcker, Constantin Bannert, and Sven Rahmann.

As the title implies, the focus of these notes is on ideas and algorithmic methods that are applied when evolutionary relationships are to be reconstructed from molecular sequences or other species-related data. Biological aspects like the molecular basis of evolution or methods of data acquisition and data preparation are not included.

Thanks

We thank Rainer Matthiesen for his notes taken during the Winter 2001/2002 lectures in Berlin, Rod Page for help with the practical course, and Heiko Schmidt for several discussions.

Contents

III Distance-based Methods

Part I

Basics

Chapter 1

Introduction: Creation, Evolution and Systematics

"Where do we come from" is one of the most frequently asked and discussed questions. Many theories and hypotheses suggest possible answers. One way to explain our origin is that we (and everything else) were created by an almighty god (Creationism). Often, creationism and evolution are presented as conflict, where people have to believe either in creation, or in evolution. However, we encourage another point of view: If we assume the existence of an almighty god, we can probably find answers to most questions. However, many of these answers depend on belief alone. In modern science, we try to answer our questions by experiments with reproducible results. This does not mean that we deny the existence of god, we just leave him/it out of consideration, because we can not prove or disprove his/its existence.

For centuries, biologists have tried to detect and classify the diversity in the biological world, this effort is known as *systematics*. If we assume that the biological diversity we see today is due to *evolution*, every species has a *phylogeny*, a history of its own evolutionary development. These two concepts gave rise to the science of *Phylogenetic Systematics*, where organisms are classified into groups by their phylogeny. Traditionally, this is based on morphological characters, while today molecular systematics prevail. Here is a short historical tour.

- Bible (*Genesis* 1, 26-27):

  > Then God said, "Let us make man in our image, in our likeness, and let them rule over the fish of the sea and the birds of the air, over the livestock, over all the earth, and over all the creatures that move along the ground." So God created man in his own image, in the image of God he created him; male and female he created them.

- Carl Linné (1707-1778): Linné revolutionized the way in which species were classified and named. He proposed to group them by shared similarities into higher taxa, being: genera, orders, classes and kingdoms. He also invented the 'binomial system' for naming species, where a name is composed of genus and species in latin, as in *Homo sapiens*. The system rapidly became the standard system for naming species. In his early years as a researcher in botany, Linné believed in invariant species. Later on, he admitted that a certain variation was possible. His most important works are *Systema naturae* (1735) and *Genera plantarum* (1737).

- Chevalier de Lamarck (1744-1829): Lamarck started as taxonomist in botany, and he was in general well educated. He applied Linné's ideas also to animals: *Philosophie zoologique* (1809). Lamarck was one of the first to believe in some kind of evolution. However, his way of explaining the observed changes was not accurate. In response to environmental changes, organisms would over- or underuse certain parts of their bodies. The heavily used parts would improve, the underused parts wither. These changes would be inherited by the offspring.

- Georges Cuvier (1769-1832): Cuvier's speciality was the comparison of organisms, characterizing their differences and similarities. He introduced the idea of the Phylum, but still believed in creation. He also studied fossils and observed changes in comparison with contemporary organisms. He explained them by a series of catastrophes. Each would have wiped out all life on earth, which would be newly created afterwards. Cuvier was a colleague of Lamarck, but he had no respect for Lamarck's theory of 'inheritance of acquired characteristics'.
- Charles Darwin (1809-1882): Here is an excerpt from the famous book *On the origin of species by means of natural selection* by Charles Darwin, 1859:

  > Whatever the cause may be of each slight difference in the offspring from their parents—and a cause for each must exist—it is the steady accumulation, through natural selection, of such differences, when beneficial to the individual, that gives rise to all the more important modifications of structure, by which the innumerable beings on the face of this earth are enabled to struggle with each other, and the best adapted to survive.

  Darwin was not the first who believed in some sort of evolution. By the time he published his famous book, many biologists did not believe in the notion of fixed species anymore. However, Darwin was the one who was able to explain *how* this evolution could have occurred. His concept of evolution by natural selection may be expressed as a very simple set of statements:

  1. The individual organisms in a population vary.
  2. They overproduce (if the available resources allow).
  3. Natural selection favors the reproduction of those individuals that are best adapted to the environment.
  4. Some of the variations are inherited to the offspring.
  5. Therefore organisms evolve.
- Ernst Haeckel (1834-1919) is known for his "genealogical tree" (1874, see Figure 1.1).

Figure 1.1: The genealogical tree by Ernst Haeckel, 1874.

Chapter 2

Graphs, Trees, and Phylogenetic Trees

2.1 Graphs

In this course we talk a lot about trees. Trees are a special kind of graph. So we start with the definitions of a few graph theoretical terms before we see how trees are defined.

Undirected Graphs.

- An *undirected graph* is a pair $G = (V, E)$ consisting of a set of *vertices* (or *nodes*) and a set $E \subset \binom{V}{2}$ of *edges* (or *branches*) that connect nodes. The number of nodes $|V|$ is also called the *size* of G.
- The set $\binom{V}{2}$ referred to above is the set of all 2-element subsets of V, i.e., the set of all $\{v_1, v_2\}$ with $v_1 \neq v_2$. Sets are used to model undirected edges because there is no order among the vertices for an undirected edge.
- If $e = (v_1, v_2) \in E$ is an edge connecting vertices v_1 and v_2, e is said to be *incident* to v_1 and v_2. The two vertices v_1 and v_2 are *adjacent*.
- The *degree* of a vertex v is the number of edges incident to v.
- A *path* is a sequence of nodes $v_1, v_2, \ldots, v_n$ where v_i and v_{i+1} are connected by an edge for all $i = 1, \ldots, n-1$. The *length* of such a path is the number of edges along the path, $n - 1$. In a *simple path* all vertices except possibly the first and last are distinct. Two vertices v_i and v_j are *connected* if there exists a path in G that starts with v_i and ends with v_j. A graph $G = (V, E)$ is *connected* if every two vertices $v_i, v_j \in V$ are connected.
- A *cycle* is a path in which the first and last vertex are the same. A *simple cycle* uses each edge $\{u, v\}$ at most once. A graph without simple cycles is called *acyclic*.

Directed Graphs.

- A *directed graph* (digraph) is a pair $G = (V, E)$, as for an undirected graph, but where now $E \subset V \times V$.
- An edge $e = (v_1, v_2)$ is interpreted to point from v_1 to v_2, symbolically also written as $v_1 \xrightarrow{e} v_2$. We usually assume that $v_1 \neq v_2$ for all edges, although the definition does allow $v_1 = v_2$; such an edge is called a *loop*. Graphs without loops are *loopless*.
- In a directed graph one distinguishes the *in-degree* and the *out-degree* of a vertex.

Additional information in graphs.

- If edges are annotated by numbers, one speaks of a *weighted graph.* Edge weights often represent lengths. The *length of a path* in a weighted graph is the sum of the edge lengths along the path.
- If edges are annotated by some label (e.g., letters or strings), one speaks of a *labeled graph.*

Graphs are useful data structures in modeling real-world problems. Applications in bioinformatics include:

- modeling metabolic, regulatory, or protein interaction networks,
- physical mapping and sequence assembly (interval graphs),
- string comparison and pattern matching (finite automata, suffix trees),
- modeling sequence space,
- phylogenetic trees,
- modeling tree space.

2.2 Trees

Definitions. A *tree* is a connected acyclic undirected graph. A *leaf* (*terminal node*) is a node of degree one. All other nodes are *internal* and have a degree of at least two. The *length of a tree* is the sum of all its edge lengths.

Let $G = (V, E)$ be an undirected graph. The following statements are equivalent.

1. G is a tree.
2. Any two vertices of G are connected by a unique simple path.
3. G is minimally connected, i.e., if any edge is removed from E, the resulting graph is disconnected.
4. G is connected and $|E| = |V| - 1$.
5. G is acyclic and $|E| = |V| - 1$.
6. G is maximally acyclic, i.e., if any edge is added to E, the resulting graph contains a cycle.

One distinguishes between rooted and unrooted trees.

- An *unrooted tree* is a tree as defined above. An unrooted tree with degree three for all internal nodes is called a *binary* tree.
- A *rooted tree* is a tree in which one of the vertices is distinguished from the others and called the *root.* Rooting a tree induces a hierarchical relationships of the nodes and creates a directed graph, since rooting implies a direction for each edge (by definition always pointing away from the root). The terms *parent*, *child*, *sibling*, *ancestor*, *descendant* are then defined in the obvious way. Rooting a tree also changes the notion of the degree of a node: The *degree of a node in a rooted tree* refers to the *out-degree* of that node according to the above described directed graph. A rooted tree with out-degree two for all internal nodes is called a *binary tree.* Each edge divides (splits) a tree into two connected components. Given a node v other than the root in a rooted tree, the *subtree rooted at v* is the remaining tree after deleting the edge that ends at v and the component containing the root. (The subtree rooted at the root is the complete, original tree.) The *depth* of node v in a rooted tree is the length of the (unique) simple path

from the root to v. The *depth of a tree T* is the maximum depth of all of Ts nodes. The *width of a tree T* is the maximal number of nodes in T with the same depth.

2.3 Phylogenetic Trees

2.3.1 What are Phylogenetic trees?

Phylogenetic (also: evolutionary) trees display the evolutionary relationships among a set of objects. Usually, those objects are species, but other entities are also possible. Here, we focus on species as objects, an example is shown in Figure 2.1.

The n contemporary species are represented by the leaves of the tree. Internal nodes are branching (out-degree two or more in a rooted tree), they represent the last common ancestor before a speciation event took place. The species at the inner nodes are usually extinct, and then the amount of data available from them is quite small. Therefore, the tree is mostly based on the data of contemporary species. It models their evolution, clearly showing how they are related via common ancestors.

Speciation. The interpretation of phylogenetic trees requires some understanding of speciation, the origin of a new species. A speciation event is always linked to a population of organisms, not to an individual. Within this population, a group of individuals emerges that is able to live in a new way, at the same time acquiring a barrier to genetic exchange with the remaining population from which it arose (see, e.g. [13]). Usually, this is due to environmental changes that lead to a spatial separation, often by chance events. After the separation of the two populations, both will diverge from each other during the course of time. Since both species evolve, the last common ancestor of the two will usually be extinct today, in the sense that the genetic pool we observe in either contemporary species is not the same as the one we would have observed in their last common ancestor.

2.3.2 Tree classification: Fully resolved vs. multifurcating

The difference between rooted and unrooted trees was already explained in Section 2.2. However, aside from being rooted or unrooted, Trees can also be *fully resolved* or not. For example in a rooted tree, each *inner* node has an in-degree of one, and an out-degree of two. The only exception is the root, which has an in-degree of zero. Trees that are not fully resolved (a.k.a. multifurcating trees) contain inner nodes with a degree of four or more. Those nodes are called unresolved nodes or

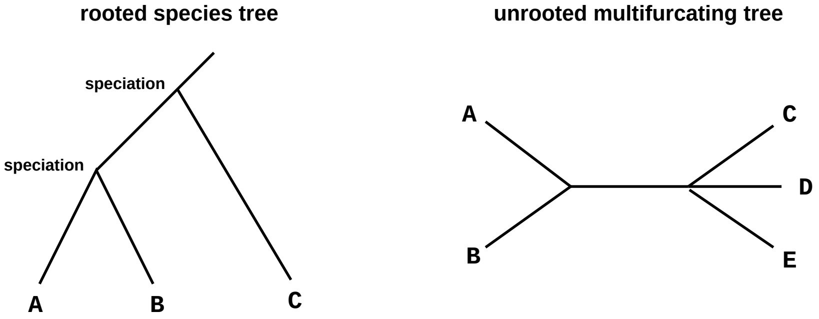


Figure 2.1: Left: Rooted, fully resolved species tree. Right: Unrooted tree with five taxa. The inner node branching to C, D and E is a polytomy.

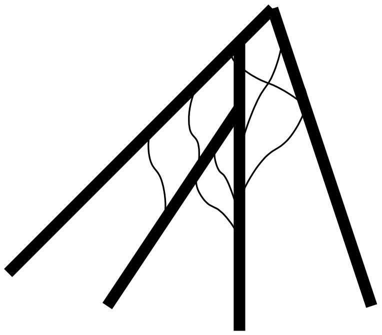
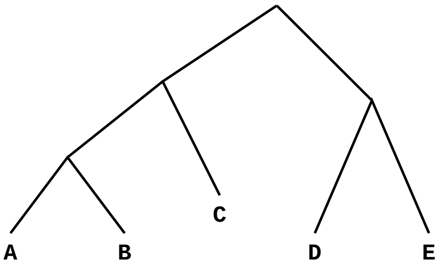


Figure 2.2: A microscopic view at evolution.



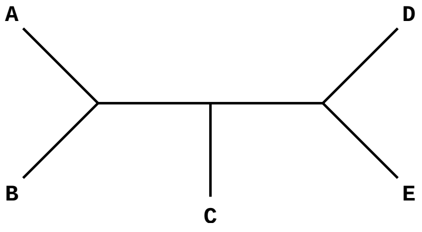
```
(((A,B),C),(D,E))
```

Figure 2.3: A rooted tree with five leaves and the corresponding parenthesis structure.

polytomies. A polytomy can be due to simultaneous speciation into multiple lineages, or to the lack of knowledge about the exact speciation order. In fact, speciation is not a punctual event anyway, see Figure 2.2.

2.3.3 Tree representation and tree shape/topology

A simple and often used notation to represent a tree is the PHYLIP or NEWICK format, where it is represented by a parenthesis structure, see Figure 2.3 (see also `http://evolution.genetics.washington.edu/phylip/newicktree.html`). This format implies a rooted tree where the root is represented by the outmost pair of parentheses. If unrooted trees are considered, different parenthesis structures can represent the same tree topology, see Figure 2.4.



```
(((A,B),C),(D,E))
((A,B),(C,(D,E)))
(A,(B,(C,(D,E))))
((A,B),C,(D,E))
etc.
```

Figure 2.4: An unrooted tree and several parenthesis structures representing this tree.

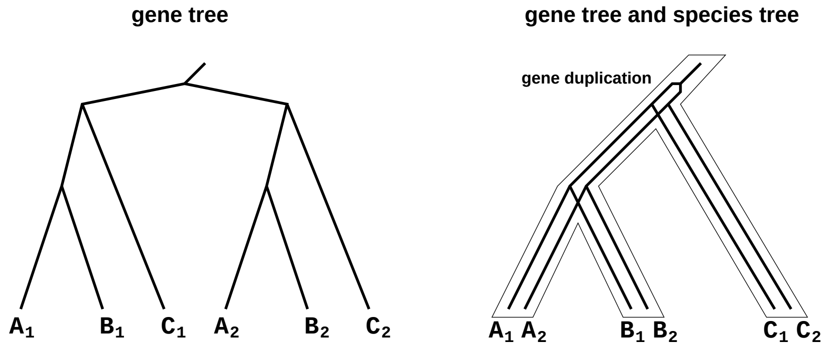


Figure 2.5: Left: Gene tree. Right: Species tree with a gene duplication.

It is important to notice that the branching order of edges at internal nodes is always arbitrary. Hence the trees $(A,(B,C))$ and $((B,C),A)$ are the same! ("Trees are like mobiles.") Sometimes, as in nature, rooted trees are drawn with their root at the bottom. More often, though, rooted trees are drawn with their root at the top or from left to right. Unrooted trees are often drawn with their leaves pointing away from the center of the picture.

2.3.4 Weights in phylogenetic trees

Often in phylogeny one has *weighted* trees, where the edge lengths represent a distance. One example is the evolutionary time that separates two nodes in the tree. Here, trees are usually rooted, the time flowing from the root to the leaves. Another example for an edge weight is the number of morphological or molecular differences between two nodes. Generally, both rooted and unrooted weighted trees are considered in phylogeny. A special class of rooted weighted trees are those where each leaf has the same distance to the root, called *dendrograms*. Note that the same tree (same topology) can represent very different weighted trees!

Unlike general graphs, trees can always be drawn in the plane. However, sometimes it is difficult or impractical to draw the edge lengths of a tree proportional to their weight. In such cases one should write the edge lengths as numbers. However, one has to make sure that the reader does not confuse these numbers with other annotations like bootstrap values. (Bootstrap values are discussed in Chapter 10.)
In modern molecular phylogeny, often the species at the leaves of a phylogenetic tree are represented by genes or stretches of genomic DNA. It might also be that more than one (orthologous) gene from the same species is included in such a *gene tree*. In a gene tree, internal nodes represent gene duplications instead of speciation events, see Figure 2.5. It is also possible that an internal node represents a *cluster* of species (the ones at the leaves in the subtree). Such a cluster is sometimes also called an *operational taxonomic unit* (OTU).

2.4 Topological Equivalence and Tree Isomorphism

We try not to puzzle the reader with the mathematical details of phylogenetics but instead, we concentrate on the ideas behind the formalism. Still and all, the reader should be aware that all informal definitions of this chapter have a formal counterpart. As an example, we will show in the following what the "topological equivalence" of two trees really is. For the used notions, see Section 2.2. We concentrate on unrooted trees here, but similar arguments apply for rooted trees.

If two trees share the same information, we cannot say that they are *equal* because this can depend on the underlying mathematical representation of a tree. Instead, we say that two trees $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$ are *isomorphic*, usually denoted $T_1 \cong T_2$, if there exists a bijection $\phi : V_1 \to V_2$ such that $\{u, v\} \in E_1$ is an edge of T_1 if and only if $\{\phi(u), \phi(v)\} \in E_2$ is an edge of T_2. Note that ϕ must satisfy both conditions at the same time, and that $\phi : V_1 \to V_2$ being a bijection implies that T_1 and T_2 have the same number of vertices.

Regarding phylogenetic trees T_1, T_2, it is not sufficient that these trees share the same topology but, in addition, the leaves of the trees must be labeled with the same taxa, respecting this topology. So, the phylogenetic trees T_1 and T_2 are *isomorphic* if there exists a bijection $\phi : V_1 \to V_2$ such that

- $\{u, v\} \in E_1$ is an edge of T_1 if and only if $\{\phi(u), \phi(v)\} \in E_2$ is an edge of T_2, and
- a vertex $v \in V_1$ is labeled x in T_1 if and only if the vertex $\phi(v) \in V_2$ is also labeled x in T_2.

Using this definitions, we can formally show that two phylogenetic trees AB—CD and AC—BD are not isomorphic, that is, have a different topology: There exist $6! = 720$ bijections between the vertex sets of these two trees. But *none* of these bijections satisfies both of the above conditions simultaneously.

2.5 Counting Trees

Finally we want to count how many different tree topologies exist for a given set of leaves. One has to distinguish between rooted and unrooted trees. Here we only consider unrooted trees.

Lemma. Given n objects, there are $U_n = \prod_{i=3}^{n}(2i-5)$ labeled unrooted trees with these objects at the leaves.

Proof. For $n = 1, 2$ recall that by definition the product of zero elements is $U_1 = U_2 = 1$. For $n \geq 3$ the proof is by induction:

Let $\boldsymbol{n = 3}$. There is exactly $U_3 = \prod_{i=3}^{3}(2i-5) = 1$ labeled unrooted tree with $n = 3$ leaves.

Let $\boldsymbol{n \geq 3}$. There are exactly n leaves, $n-2$ internal nodes and $2n-3$ edges. A new edge (creating the $(n+1)$-th leaf) can be inserted in any of the $2n-3$ edges. Hence, by induction hypothesis,

$$\begin{aligned} U_{n+1} &= U_n \cdot (2n-3) \\ &= \prod_{i=3}^{n}(2i-5) \cdot (2(n+1)-5) \\ &= \prod_{i=3}^{n+1}(2i-5). \end{aligned}$$

$\square$

Hence the number of unrooted labeled trees grows exponentially with the number of leaves n. This is what makes life difficult in phylogenetic reconstruction. The number of unresolved (possibly multifurcating) trees is even larger.

Chapter 3

Characters, States, and Perfect Phylogenies

3.1 Characters and States

Given a group of species and no information about their evolution. How can we find out the evolutionary relationships among them? We need to find certain properties of these species, where the following must hold:

- We can decide if a species has this property or not.
- We can measure the quality or quantity of the property (e.g., size, number, colour).

These properties are called *characters*. The actual quality or quantity of a character is called its *state*.

Definition. A *character* is a pair (λ, C) consisting of a property name λ and an arbitrary set C, where the elements of C are called *character states*.

Examples:

- The existence of a nervous system is a binary character. Character states are elements of $\{False, True\}$ or $\{0, 1\}$.
- The number of extremities (arms, legs,...) is a numerical character. Character states are elements of $\mathbf{N}$.
- Here is an alignment of DNA sequences:
  Seq1: A C C G G T A
  Seq2: A G C G T T A
  Seq3: A C T G G T C
  Seq4: T G C G G A C
  A nucleotide in a position of the alignment is a character. The character states are elements of {A,C,G,T}. For the above alignment, there are seven characters (columns of the alignment) and four character states.

The definition given for characters and states is not restricted to species. Any *object* can be defined by its characters. It may be considered as a vector of characters.

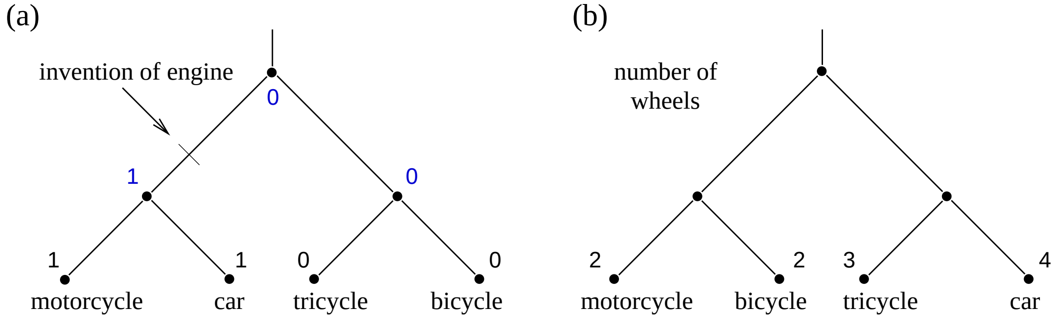


Figure 3.1: Putative phylogenies of vehicles. Tree (a) is compatible with the character 'existence of engine'. The engine is invented once, in the edge connecting the root and the common ancestor of car and motorcycle. Tree (b) is compatible with 'number of wheels'.

.

Example: Bicycle, motorcycle, tricycle and car are objects. The number of wheels and the existence of an engine are characters of these objects. The following table holds the character states:

	# wheels	existence of engine
bicycle	2	0
motorcycle	2	1
car	4	1
tricycle	3	0

3.2 Compatibility

The terms and definitions of characters and states originate from paleontology. A main goal of paleontolgists is to find correct phylogenetic trees for the species under consideration. It is generally agreed that the invention of a 'new' character state is a rare evolutionary event. Therefore, trees where the same state is invented multiple times independently are considered less likely in comparison with trees where each state is only invented once.

Definition: A character is *compatible* with a tree if all nodes of the tree can be labeled such that each character state induces one connected subtree.

Note: This implies that a character with k states in our data will be compatible with a tree if we observe exactly $k - 1$ changes of state in that tree.

Example: Given a phylogenetic tree on a set of objects and a binary character $c = \{0, 1\}$: If the tree can be divided into two subtrees, where all nodes on one side have state 0, and 1 on the other, we count only one change of state. This can be seen in Figure 3.1 (a). The character 'existence of engine' is compatible with the tree, as the motor is invented only once (if the inner nodes are labeled correctly). The same character is not compatible with the tree in Figure 3.1 (b), where the engine is invented twice (if we assume that the common ancestor had no engine). The character 'number of wheels' is compatible with both trees.

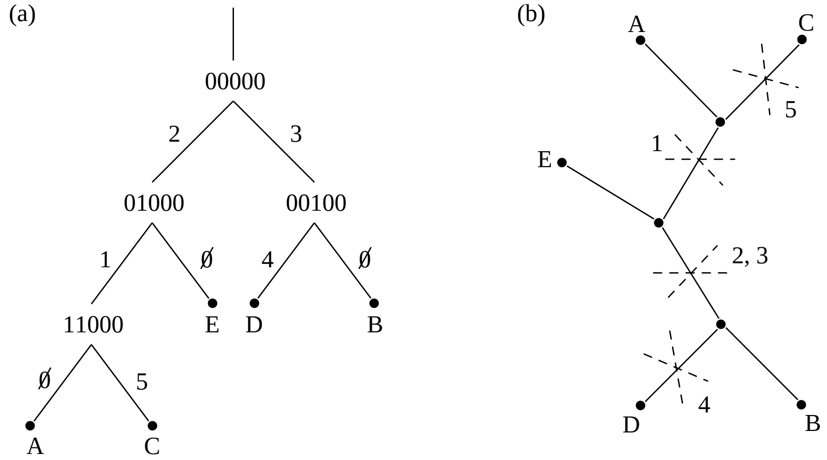


Figure 3.2: Perfect phylogeny for binary character states of M shown as (a) a rooted tree (numbers at edges denote invention of characters), and (b) an unrooted tree (crossed dashed lines represent splits, a number at a split denotes the character inducing the split)

Exercise: The objects A, B, C, D share three characters 1,2,3. The following matrix holds their states:

	1	2	3
A	a	α	d
B	a	β	e
C	b	β	f
D	b	γ	d

Look at all possible tree topologies. Is there a tree that all characters are compatible with?

3.3 Perfect Phylogenies

Let a set $\mathcal{C}$ of characters and a set S of objects be given.

Definition: A tree T is called a *perfect phylogeny (PP)* for $\mathcal{C}$ if all characters $C \in \mathcal{C}$ are compatible with T.

Example: The objects A, B, C, D, E share five binary (two-state) characters. The matrix M holds their binary states:

	1	2	3	4	5
A	1	1	0	0	0
B	0	0	1	0	0
C	1	1	0	0	1
D	0	0	1	1	0
E	0	1	0	0	0

The trees in Figure 3.2 show a perfect phylogeny for M. There are two ways of looking at this perfect phylogeny:

- As a rooted tree in a directed framework with development (see Figure 3.2 (a)): A (virtual) ancestral object is added as the root. Its characters have state 00000. During development, on the way from this root to the leaves, a character may invent a new state '1' exactly once. In the subtree below the point where the state switches, the state will always remain '1'.
- As an unrooted tree in a split framework (see Figure 3.2 (b)): A binary character corresponds to a split (a bipartition) in a set of objects. In a tree this is represented by an edge. In this framework, characters 4 and 5 are *uninformative*. The splits they induce are trivial as they separate a single object from the other ones and hence do not tell us about the relationships among the objects. Character 3 is just the opposite of character 2. It describes the same split.

We can define an *object set* for each character: character i corresponds to the set of objects O_i where the character i is "on", that is it has the value 1. Then each column of the matrix M corresponds to a set of objects: $O_1 = \{A, C\}, O_2 = \{A, C, E\}, O_3 = \{B, D\}, O_4 = \{B\}, O_5 = \{C\}$.

The *Perfect Phylogeny Problem (PPP)* addresses the question if for a given matrix M there exists a tree as shown in Figure 3.2 (a) and, given its existence, how to construct it.

An alternative formulation for the PPP with binary characters is the formulation as a *Character Compatibility Problem*: Given a finite set S and a set of splits (binary characters) (A_i, B_i) such that $A_i \cap B_i = \emptyset$ and $A_i \cup B_i = S$, is there a tree as in Figure 3.2 (b) that realizes these splits?

In general, the PPP is NP-hard. Given binary characters, it is easy, though. Gusfield [14] formulates the following theorem:

Theorem. *M has a PP if and only if for any 2 columns i, j there holds one of:*

(i) $O_i \subseteq O_j$

(ii) $O_j \subseteq O_i$

(iii) $O_i \cap O_j = \emptyset$

The condition in the above theorem describes the compatibility of two characters. Two characters are compatible if they allow for a PP. And Gusfield's theorem becomes: "A set of binary characters has a PP if all pairs of characters are compatible." (Note: This is not true in general, i.e. for r-state characters with $r > 2$, as there are many "tree realizations"; see [26])

A simple algorithm to test for the existence of a PP is to test all pairs of columns for the above conditions which has time complexity $O(nm^2)$, where n is the number of rows in M and m is the number of columns in M. Gusfield [14] presents an algorithm with $O(nm)$ time complexity for recognition and construction of a PP.

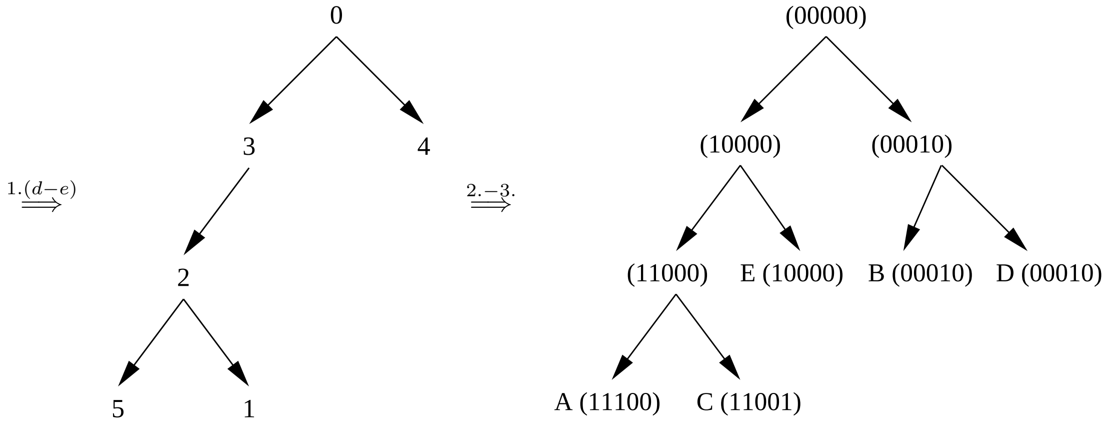
1. Draw the Hasse diagram, starting with the full set:
    a) apply radix sort to the columns of matrix M;
    b) delete columns that are identical with their right neighbor;
    c) add a column number 0 containing only ones;
    d) create a graph with a node for each column of the new matrix;
    e) note that when $O_j \subset O_i$, then $j > i$; draw an edge (i, j) between direct successors in the partial order defined by the O_i-set-inclusion. Because of the set inclusion / disjointness conditions, the obtained graph will form a tree.

2. Annotate Each node (object set) with the intersection of its rows.
3. Add leaves for single-object sets where necessary.
4. Annotate each edge by the character (if any) that is invented (becomes 1) along that edge.

Another $O(nm)$ time algorithm can be found in [15], Chapter 17.3.4.

Example:

$$\begin{array}{c|ccccc}
 & 1 & 2 & 3 & 4 & 5 \\ \hline
A & 0 & 1 & 1 & 0 & 1 \\
B & 0 & 0 & 0 & 1 & 0 \\
C & 1 & 1 & 1 & 0 & 0 \\
D & 0 & 0 & 0 & 1 & 0 \\
E & 0 & 0 & 1 & 0 & 0
\end{array}
\quad \overset{1.\text{(a)-(c)}}{\Longrightarrow} \quad
\begin{array}{c|cccccc}
 & 0 & 3 & 2 & 5 & 4 & 1 \\ \hline
A & 1 & 1 & 1 & 1 & 0 & 0 \\
B & 1 & 0 & 0 & 0 & 1 & 0 \\
C & 1 & 1 & 1 & 0 & 0 & 1 \\
D & 1 & 0 & 0 & 0 & 1 & 0 \\
E & 1 & 1 & 0 & 0 & 0 & 0
\end{array}$$



Part II

Parsimony

Chapter 4

The Small Parsimony Problem

4.1 Introduction

There are many possibilities to construct a phylogenetic tree from a set of objects. Of course, we want to find the 'best' tree, or at least a good one. Judging the quality of a phylogentic tree requires criteria. *Parsimony* is such a criterion.

The general idea is to find the tree(s) with the minimum amount of evolution, i.e., with the fewest number of evolutionary events. We call this tree the *most parsimonious tree.* If we can find a perfect phylogeny, we can be sure that we have found an optimal result. However, most 'real world' data sets have no PP.

There are several methods to reconstruct the most parsimonious tree from a set of data. First, they have to find a possible tree. Second, they have to be able to calculate and optimize the changes of state that are needed. As the second problem is the easier subproblem of the general parsimony problem, we start with this one: We assume that we are given a tree topology and character states at the leaves, and the problem is to find a most parsimonious assignment of character states to internal nodes. This problem is referred to as the *small parsimony problem.*

We present two algorithms: A dynamic programming algorithm due to Sankoff that works for general state change costs, and a special version due to Fitch that assumes unit costs.

4.2 The Sankoff Dynamic Programming Algorithm

We assume that a fixed tree topology is given with n aligned sequences at its leaves. We assume further that a character change from x to x' has a cost that is given as a function $cost(x, x')$. Often this will be given by *unit costs* $cost(x, x') = 0$ for $x = x'$ and $cost(x, x') = 1$ for $x \neq x'$. Since the sequences are aligned, we can treat the columns of the alignment separately and in the following will focus on a single column. The formal version of the problem is then the following.

The minimum mutation problem for a single position. Given a rooted or unrooted tree with n leaves (not necessarily binary) and a single character labeling each leaf, the *minimum mutation problem* is to label each internal node of the tree with a single character so as to minimize the number of edges whose endpoints have different labels.

The algorithm is as follows (see Figure 4.1 for an example).

1. If the tree is unrooted, the root can be placed anywhere.

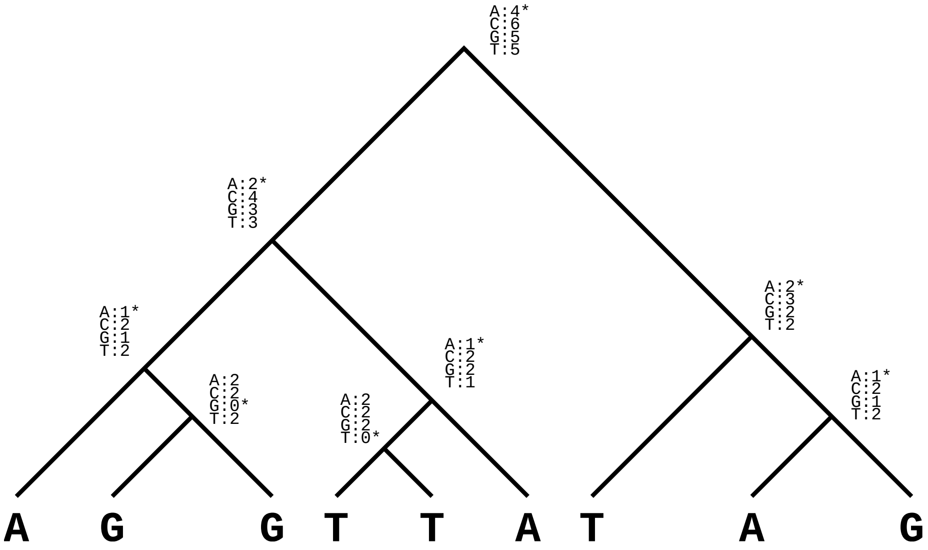


Figure 4.1: The dynamic programming algorithm with unit costs.

2. The algorithm is a dynamic programming algorithm that traverses the tree bottom-up from the leaves to the root in a way such that when a node is processed, all its children have already been processed. Obviously the root is the last node processed by this traversal.

   During this traversal, assume we process a node v. Define $C(v)$ as the cost of the optimal solution of the minimum mutation problem for the subtree rooted at v. Let $C(v,x)$ be the cost of the best labeling of subtree T_v when node v is required to be labeled with character x. Obviously, $C(v) = \min_x C(v,x)$.

   2a. The base cases for $C(v,x)$ are $C(v,x) = 0$ if leaf v is labeled with character x and $C(v,x) = \infty$, otherwise.

   2b. The recurrence relation to compute $C(v,x)$ for internal nodes is

$$C(v,x) = \sum_{v' \text{ child of } v} \min_{x} \left[C(v',x') + cost(x,x')\right].$$

3. The optimal assignment of characters to the internal nodes is then obtained in a backtracking phase, as usual in dynamic programming.

   3a. The root r is assigned a character x_r such that $C(r) = C(r, x_r)$.

   3b. In a top-down traversal then the child v' of a node v is assigned the character x_v that yielded the maximum in the bottom-up pass.

It is easy to see that this algorithm solves the minimum mutation problem correctly in $O(n\sigma^2)$ time and space, where σ is the alphabet size. For unit costs, a speedup to $O(n\sigma)$ is possible; see also the next section.

4.3 The Original Fitch Algorithm

The original description of the algorithm by Fitch (1971), analysed by Hartigan (1973) aassumes unit costs. A careful comparison with the dynamic programming algorithm above will reveal the equivalence. It is probably easiest to compare it to argue that the state set defined below equals the set of characters x that have a minimal $C(v, x)$.

The algorithm is as follows (see Figure 4.2 for an example).

For each node v of the tree we construct its *state set* S_v in the following recursive way.

1. For each leaf v, set $S_v = \{x\}$ if x is the character at that leaf.
2. The state sets of the internal nodes are computed in a bottom-up pass of the tree. Assume an internal node v with children u and w.
   a. If the intersection of the state sets S_u and S_w is non-empty, then S_v becomes this intersection.
   b. Otherwise S_v becomes the union of its children's state sets.

   More formally:

$$S_v = \begin{cases} S_u \cap S_w & \text{if } S_u \cap S_w \neq \emptyset \\ S_u \cup S_w & \text{otherwise.} \end{cases}$$

The most parsimonious reconstruction of characters at the internal nodes is then obtained in a top-down pass according to the following rules:

1. If the state set of the root contains more than one element, arbitrarily assign any of these characters to the root.
2. Let u be a child of node v, and let x_v denote the character assigned to v.
   a. If x_v is contained in S_u, assign it to node u as well.
   b. Otherwise, arbitrarily assign any character from S_u to node u.

The length of the most parsimonious tree can then easily be read from the character assignments in the tree.

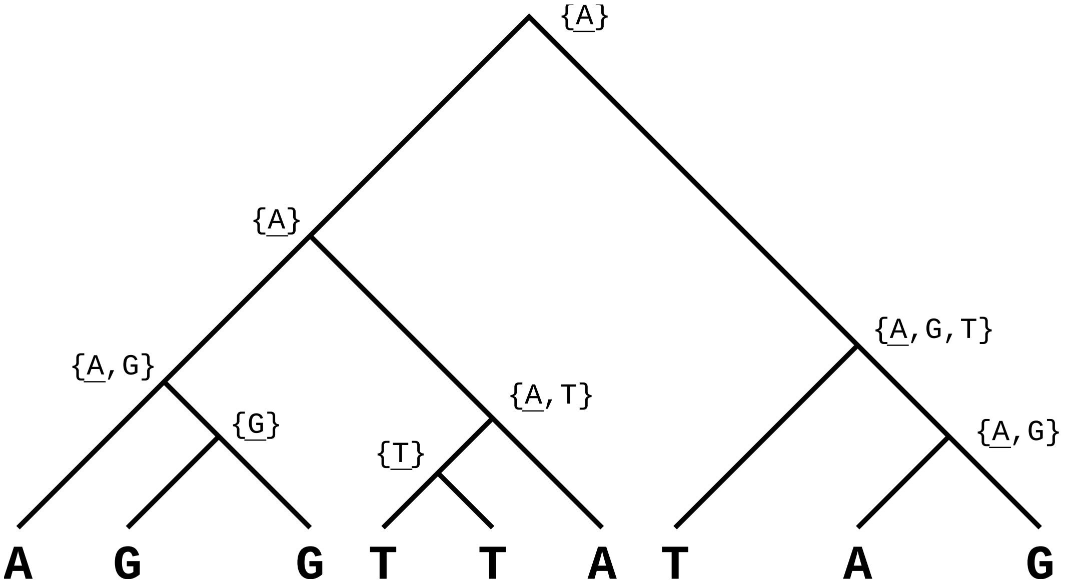


Figure 4.2: The original version of the Fitch algorithm.

Chapter 5

Maximum Parsimony

5.1 Introduction

The algorithms covered in the previous chapter solve the ‘small parsimony problem’, where a tree is given. They compute the most parsimonious assignment of states to the inner nodes of the tree. However, if the shape of the given tree was not optimal, the small parsimony solution may be not optimal among all possible most parsimonious trees.

Therefore, the algorithms that solve the *Maximum Parsimony Problem* find the optimal tree among all possible ones. We present some methods that solve the problem in this chapter. One problem of the parsimony approach is shown in Section 5.5.

5.2 Exploring the Tree Space

In principle one can solve the small parsimony problem for all possible tree topologies, and the shortest one will be the most parsimonious tree. This method will yield the optimal, i.e. most parsimonious tree. However, we have seen that the number of possible tree topologies grows extremely quick with the number of taxa. An exhaustive enumeration is in practice infeasible for more than about 12 taxa. Therefore, we will discuss several methods that allow to speed up this search in many cases.

5.2.1 Branch and Bound Heuristics.

Heuristic methods In general, one speaks of a *heuristic* method for the solution of a problem if the method uses exploratory problem-solving techniques based on experience or trial-and-error, without in the worst case necessarily to perform better or faster than naive methods. One speaks of a *Branch-and-Bound* heuristic whenever it is possible to restrict the search such that not the complete theoretically possible search space has to be examined, but one can already stop whenever a partial solution provably can not lead to an (optimal) solution of the complete problem. The maximum parsimony problem is well suited to apply the branch and bound heuristic.

There are two alternative ways to apply branch and bound heuristics to the problem of finding the most parsimonious tree. Assume an alignment with m columns is given, as shown in Figure 5.1. The first strategy does branch and bound on the alignment columns, and the second one does branch and bound on the alignment rows, i.e. the leaves of the tree. Both procedures have in common that they start with an upper bound on the length of the most parsimonious tree, obtained e.g. by an approximation algorithm such as a minimum spanning tree (see 5.3.3).

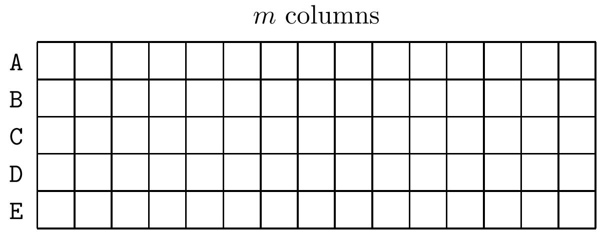


Figure 5.1: A multiple alignment for five taxa `A`, `B`, `C`, `D`, `E` with m columns.

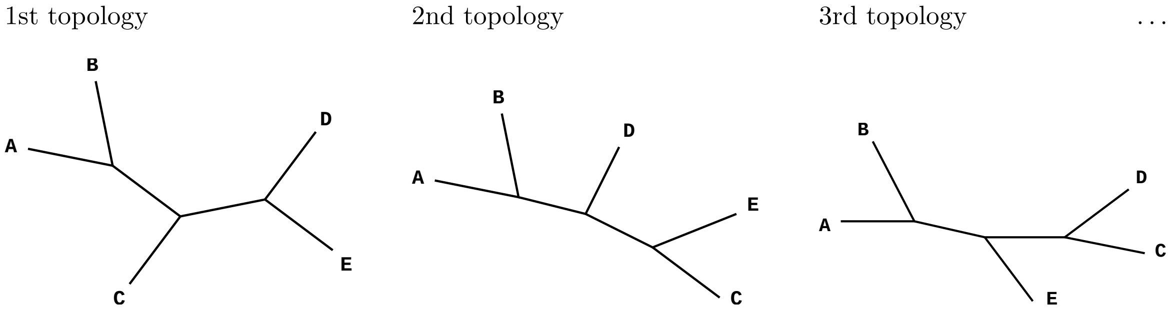


Figure 5.2: Branch and bound applied to the alignment columns.

Column-wise Branch-and-Bound

Intuitively it seems reasonable to apply branch and bound to the columns of the sequence alignment, since they are independent by assumption. For each possible tree topology, one would first compute the minimal length for the first alignment column, then for the second, and so on. Once the sum of these lengths is larger than the best solution computed so far, the procedure stops and continues with the next tree topology (see Figure 5.2). The speedup by this procedure is not impressive, though.

Row-wise Branch-and-Bound

The second approach, which is due to Hendy and Penny (1982), does branch and bound on the sequences, in parallel to the enumeration of all possible tree topologies. The idea is that adding branches to a tree can only increase its length. Hence we start with the first three taxa, build the only possible unrooted tree for them and compute its length. Then we add the next taxon in the three possible ways, thereby generating the three possible unrooted trees for four taxa. Whenever one of the trees already has length larger than the best solution computed so far, the procedure stops, otherwise it is continued by adding the next taxon, etc. (see Figure 5.3). This procedure may be used for finding the maximum parsimony tree for up to 20 taxa. The actual speedup that can be achieved depends on the input alignment, however. In a worst case scenario, the actual speedup may be quite small.

For even larger numbers of taxa, heuristic procedures have to be used. Usually, from an initial good guess one tries to improve the tree by local modifications including *branch swapping* and *subtree pruning and regrafting.*

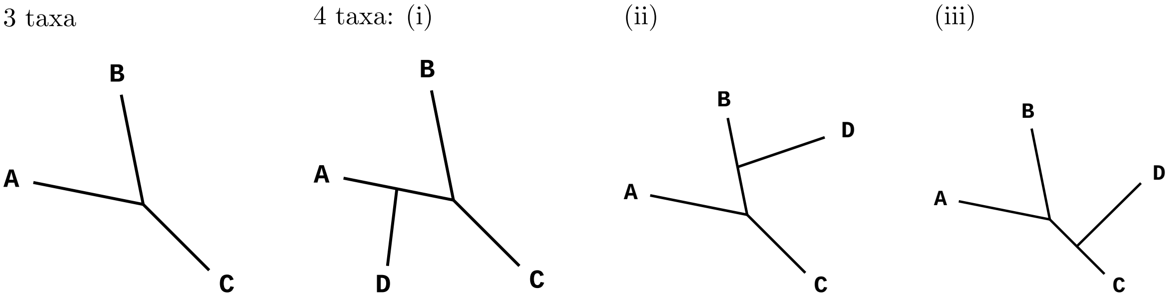


Figure 5.3: Branch and bound applied to the alignment rows.

5.2.2 Sequential Addition Greedy

Adding a taxon in all possible edges of an existing tree, thereby generating a number of new trees, is called *sequential addition*. This approach is used to ennumerate the tree space in the row-wise Branch-and-Bound scenario in Section 5.2.1. The idea can be used in an even faster, but less accurate greedy approach: Starting with a tree with three taxa, a fourth taxon is added to each possible edge, and the net amount of evolutionary change is determined (e.g., by using the Fitch algorithm). However, only the best of the resulting trees is kept. It is used as input in the next iteration.

The best tree computed in this way is not necessarily the most parsimonious one, but mostly a good approximation. The quality of the resulting tree also depends on the choice of the three taxa that are used for the initial tree [10].

5.3 Steiner Trees and Spanning trees

Steiner trees can also be used to solve the Maximum Parsimony Problem. However, finding the best Steiner tree is itself a difficult problem. Before we explain it (see 5.3.5), we introduce some basic concepts.

5.3.1 The DNA Grid Graph

If the objects whose phylogenetic tree we wish to reconstruct are DNA sequences, the *DNA grid graph* provides the means to formalize the notion of such a tree “as perfect as possible”.

Definition. The *DNA grid graph* is built as follows. Let the nodes of the graph be *all* DNA sequences of length n. Draw an edge whenever there is exactly one mismatch between two nodes (sequences).

In this graph, the length of a shortest path between two nodes is the *Hamming distance* between the represented sequences.

Now suppose that we are given N aligned sequences of length n. These define the set of terminal nodes U in this graph.

5.3.2 Steiner Trees

Definition. Given a connected weighted graph $G = (V, E)$ and a subset of the vertices $U \subseteq V$ of *terminal nodes*. A tree that is a subgraph of G and connects the vertices in U is called a *Steiner tree* of U.

Note that in general the Steiner tree will also contain a subset of the remaining vertices of G, not in U.

Definition. A Steiner tree of minimal length is a *minimum Steiner tree*.

Minimum Steiner Tree Problem. Given a connected weighted graph $G = (V, E)$ and a subset of the vertices $U \subseteq V$, find a minimum Steiner tree of U.

Observation. Assume once again that the set of terminal vertices U are objects from the DNA grid graph. Then, the minimum Steiner tree of U will be a shortest tree that connects all vertices in U, and hence, explains the DNA sequence data.

Theorem. The Minimum Steiner Tree Problem is NP complete [12].

5.3.3 Spanning Trees

Definition. Let $G = (V, E)$ be a connected weighted graph. A tree that connects all vertices of G is called a *spanning tree*. A spanning tree of minimum length is called a *minimum spanning tree*.

Property. Let T be a minimum spanning tree in a graph G. Let e be an edge in T, splitting T into two subtrees T_1 and T_2. Then e is of least weight among all the edges that connect a node of T_1 and a node of T_2.

There are fairly simple algorithms to construct a minimal spanning tree. Two such algorithms are presented in Section 24.2 of [6], Kruskal's algorithm and Prim's algorithm. Both algorithms run in $O(|E| \log |V|)$ time. They are based on the following *generic algorithm*:

Given a graph G, maintain a set of edges A that in the end will form the minimum spanning tree. Initially, A is empty. Then step by step *safe* edges are added to A, in the sense that an edge (u, v) is safe for A if $A \cup \{(u, v)\}$ is a subset of a minimum spanning tree.

Kruskal's Algorithm implements the selection of a safe edge as follows: Find, among all edges that connect two trees in the growing forest that one of least weight.

Correctness: Follows from the above property of minimum spanning trees.

Implementation:

1. Sort the edges in E by nondecreasing weight.
2. For each edge (u, v) in this order, test if it forms a circle (for example by maintaining for each node a representative element from the connected component that it is contained in), and if not, add it to A.

The first step takes $O(|E| \log |E|)$ time, the second one can be implemented in $O(|E|\alpha(|E|, |V|))$ time using a disjoint-set data structure where α is the (very slowly growing) inverse of Ackermann's function.

Prim's Algorithm Prim's algorithm also follows the generic algorithm, using a simple greedy strategy. The selected edges always form a single tree. The tree starts from an arbitrary root vertex and at each step an edge of minimum weight connecting a vertex from the tree with a non-tree vertex is selected. This is repeated until the tree spans all vertices in V.

Correctness: Follows from the above property of minimum spanning trees.

Implementation: Maintain a priority queue that contains all vertices that are not yet members of the tree, based on the least weight edge that connects a vertex to the tree. Using a binary heap, the priority queue can be implemented in $O(|V| \log |V|)$ time, resulting in a total running time of $O(|E| \log |V|)$. (This can be improved to $O(|E| + |V| \log |V|)$ by using a Fibonacci heap.)

5.3.4 Spanning Trees as Approximations of Steiner Trees

Definition. A *constant factor approximation algorithm* is a heuristic algorithm for the solution of a problem such that the result of the heuristic will deviate from the optimal solution by at most a certain multiplicative factor.

For example, a 2-approximation of a minimizing optimisation problem is an algorithm whose solution is guaranteed to be at most twice the optimal solution (an example is the spanning tree approximation to the classic Traveling Salesman problem).

A simple approximation algorithm to compute a Steiner tree on a set of vertices U is the *spanning tree heuristic*:

1. Compute all-pairs shortest-paths on the terminal nodes in U. Complexity: $O(|V|(|V| + |E|))$.
2. Compute a minimum spanning tree on this complete, weighted subgraph. Complexity: $O(|U|^2 \log |U|)$.
3. Map back the minimum spanning tree into the original graph. Complexity: $O(|E|)$.

Theorem. The length of the resulting Steiner tree is at most twice the length of the minimum Steiner tree.

5.3.5 Application to phylogeny

Summary: Given a set of DNA sequences whose phylogeny we want to reconstruct, we represent them as vertices in a DNA grid graph. Now we can use the spanning tree heuristic to compute a Steiner tree on these vertices. This tree is a 2-approximation of the correct Steiner tree, i.e., one with the minimum amount of evolutionary changes. BUT: The approximated Steiner tree is not necessarily binary, nor will it necessarily have the terminal nodes as leaves (which would be expected of a phylogenetic tree).

5.4 Generalized Tree Alignment

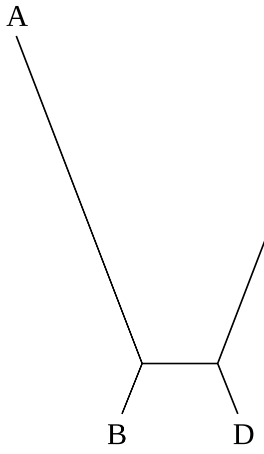
In all of the above discussion, we assumed that a multiple alignment of the sequences was given. If that is not the case, we have an even harder problem, the

Generalized Tree Alignment Problem: Given k sequences, find a tree with the sequences at the leaves and new reconstructed sequences at the internal nodes such that the length of the tree is minimal. Here the length of the tree is the sum of the edit distances of the sequences at the end of the edges.

This is a very hard problem. A dynamic programming algorithm that runs in exponential time exists [20]. Heuristic approximations also exist, see e.g. [25] and [21].

5.5 Inconsistency of Maximum Parsimony

A standard objection to the parsimony criterion is that it is not *consistent*, i.e. even with infinite amount of data (inifinitely long sequences), one will not obtain the correct tree. The standard example for this behavior is subsumed under the term *long branch attraction*. Assume the following tree with correct topology $((A, B), (C, D))$.



Since the branches pointing to A and C are very long, each of these two taxa looks essentially random when compared to the other three taxa. The only pair that does not look random is the pair (B, D), and hence in a most parsimonious tree these two will be placed together giving the topology $((B, D), (A, C))$.

Part III

Distance-based Methods

Chapter 6

Distance Based Trees

Distance based tree building methods rely on a distance measure between sequences resulting in a distance matrix. Distance measures usually take a multiple alignment of the sequences as input. After the distance measure is performed sequence information is not used any more. This is in contrast to character based tree building methods which consider each column of a multiple sequence alignment as a character and which assess the nucleotides or amino acid residues at those sites (the character states) directly.

The idea when using distance based tree building methods is that knowledge of the "true evolutionary distances" between homologous sequences should enable us to reconstruct their evolutionary history.

Suppose the evolutionary distances between members of a sequence set $\{A, B, C, D, E\}$ were given by a distance matrix d^M:

	A	*B*	*C*	*D*	*E*
A	0	200	300	600	600
B		0	300	600	600
C			0	600	600
D				0	200
E					0

For example, the value 300 in the first row of the above matrix shall be read as "the evolutionary distance between A and C is 300."

Consider now the tree in Figure 6.1. A tree like this is called a *dendrogram* as the nodes are ranked on the basis of their relative distance to the root. The amount of evolution which has accumulated in A and C since divergence from their common ancestor is 150. In other words, the evolutionary distance from A (and C) to the common ancestor of A and C is 150. In general, the sum of edge weights along the path between two nodes corresponds to the evolutionary distance between the two nodes. Deriving distances between leaves is done by summing up edge weights along the path between the leaves. Distances derived in this way from a tree form the *path metric* d^T of the tree. For the tree in Figure 6.1, we see that $d^T = d^M$.

6.1 Basic Definitions

Definition A *metric* on a set of objects O is given by an assignment of a real number d_{ij} (a distance) to each pair $i, j \in O$, where d_{ij} fulfills the following requirements:

(i) $d_{ij} > 0 \quad for\ i \neq j$

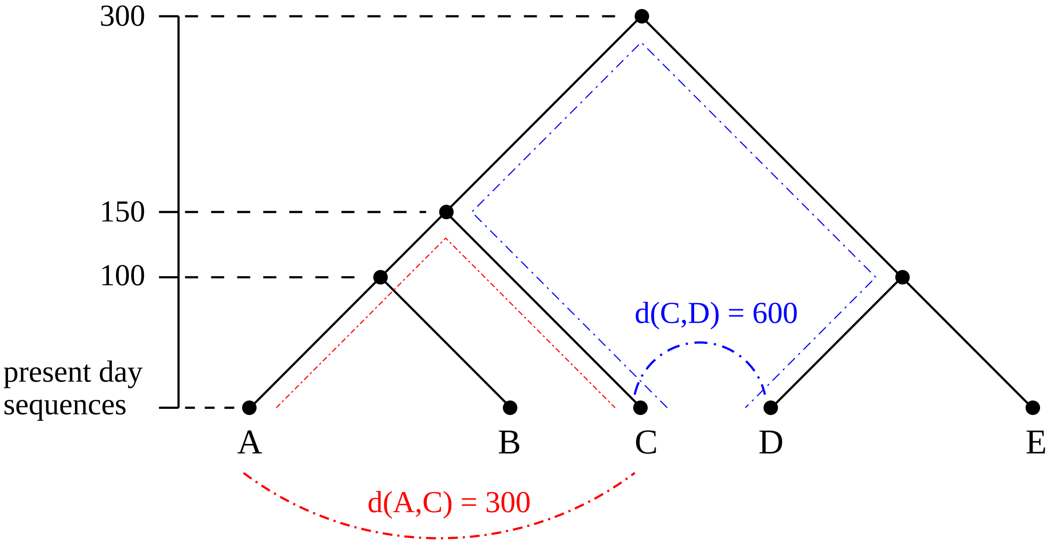


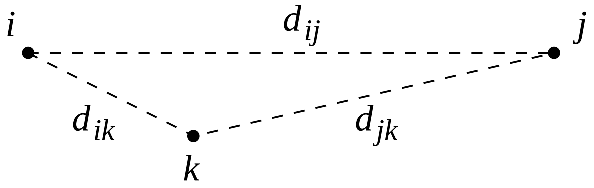
Figure 6.1: Rooted phylogenetic tree with weighted edges. A root has been assigned assuming that the sequences have evolved from a common ancestor. The sum of edge weights along the path between two leaves is the evolutionary distance between the two leaves. These distances d^T correspond to the ones measured and held in d^M.

(ii) $d_{ij} = 0 \quad for\ i = j$

(iii) $d_{ij} = d_{ji} \quad \forall\ i, j \in O$

(iv) $d_{ij} \leq d_{ik} + d_{kj} \quad \forall\ i, j, k \in O$

The latter requirement is called the *triangle inequality*:



Definition Let d be a metric on O. d is an *additive metric* if it satisfies

$$d_{ij} + d_{kl} \leq max(d_{ik} + d_{jl}, d_{il} + d_{jk}) \quad \forall\ i, j, k, l \in O.$$

An alternative and equivalent formulation is the *four point condition* of Buneman [5]:

Four point condition: d is an *additive metric* on O, if any four elements from O can be named x, y, u and v such that

$$d_{xy} + d_{uv} \ \leq \ d_{xu} + d_{yv} \ = \ d_{xv} + d_{yu}.$$

See Figure 6.2.

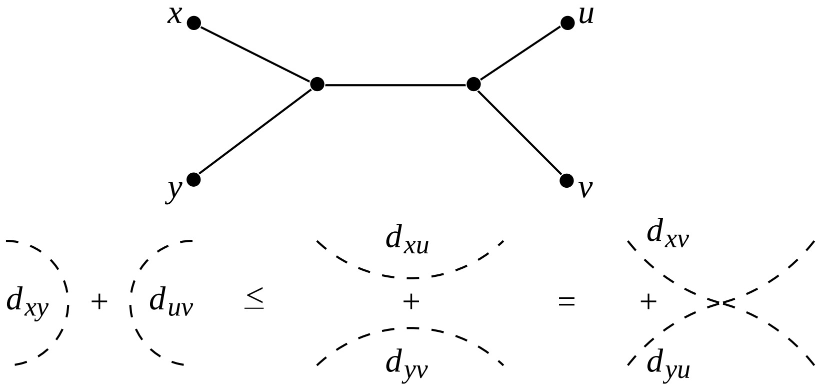


Figure 6.2: The four point condition is a strengthened version of the triangle inequality. It implies that the path metric of a tree is an additive metric.

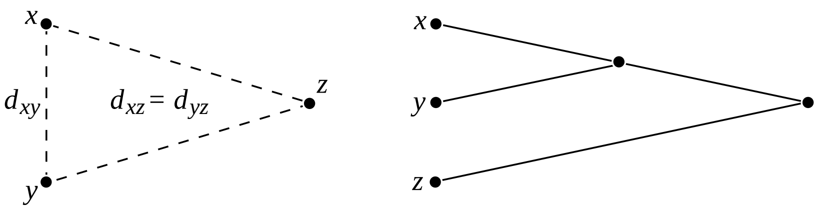


Figure 6.3: Three point condition

Definition d is an *ultrametric* if it satisfies

$$d_{ij} \leq max(d_{ik}, d_{jk}) \quad \forall \; i, j, k \in O.$$

There is again an alternative formulation.

Three point condition: d is an *ultrametric* on O, if any three elements from O can be named x, y, z such that

$$d_{xy} \; \leq \; d_{xz} \; = \; d_{yz}.$$

See Figure 6.3.

This is an even stronger version of the triangle inequality. If d is an ultrametric, it is an additive metric.

6.2 Ultrametric Trees

A weighted tree is called an *ultrametric tree* if it can be rooted in such a way that the distance from the root to any leaf is equal. The dendrogam of Figure 6.1 is an ultrametric tree. There is a clear interpretation inherent to ultrametric trees: Sequences have evolved from a common ancestor at constant rate (molecular clock hypothesis).

The path metric of an ultrametric tree is an ultrametric. Conversely, if distances d^M between a set of objects form an ultrametric, there is one ultrametric tree T corresponding to the distance measure, that is $d^T = d^M$. Given an ultrametric, this ultrametric tree can easily be reconstructed by one of the agglomerative clustering procedures described below.

Distance measures on real sequence data generally don't form an ultrametric. However, if the observed distances are close to an ultrametric, clustering procedures such as UPGMA (see Section 6.2.1) are the simplest and fastest way to reconstruct an ultrametric tree. While this is a very common approach, it is a heuristic ("algorithmic") method which does not optimize a simple objective function. As mentioned above, being close to an ultrametric implies the existence of a molecular clock or a constant rate of evolution (which sometimes may hold for closely related sequences). Note that clustering procedures are sensitive to unequal evolutionary rates. If the assumption of rate constancy among lineages does not hold, UPGMA may give the wrong topology (for an example see Section 6.3.5 and Figure 6.4).

Other distance based methods like Neighbor Joining are more general, that is, they do not presume a molecular clock (see Section 6.3.5).

6.2.1 Agglomerative Clustering

Agglomerative clustering is conceptually simple and fast. Singleton clusters are successively merged into larger clusters to form a hierarchy:

Given a set of objects O with n elements and distances $d_{i,j},\ i, j \in O$, initially each object is assigned a singleton cluster. Then the algorithm proceeds as follows:

While there is more than one cluster left, do:

1. Find the pair (i, j) with the smallest distance d_{ij}.
2. Create a new cluster u that joins clusters i and j.
3. Define the *height* (i.e. distance from leaves) of u to be $l_{ij} := d_{ij}/2$
4. Compute the distance d_{ku} of u to any other cluster $k \notin \{i, j\}$ in one of the ways described below.
5. Remove i, j from the list of objects

Different clustering methods differ in how they define the distance d_{ku} between two clusters in Step 4:

single linkage clustering:

$$d_{ku} := \min(d_{ki}, d_{kj})$$

complete linkage clustering:

$$d_{ku} := \max(d_{ki}, d_{kj})$$

UPGMA (unweighted pair group method using arithmetic averages):

$$d_{ku} := \frac{n_i d_{ki} + n_j d_{kj}}{n_i + n_j}$$

where n_i is the number of elements in cluster i. d_{ku} is the arithmetic average of the original distances of all elements in k and all elements in u.

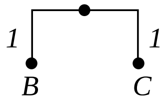
WPGMA (weighted pair group method using arithmetic averages):

$$d_{ku} := \frac{d_{ki} + d_{kj}}{2}$$

Example Given a set of objects $O = \{A, B, C, D, E\}$ and an ultrametric distance matrix d^M on O with entries

	A	B	C	D	E
A	0	8	8	12	8
B		0	2	12	4
C			0	12	4
D				0	12
E					0

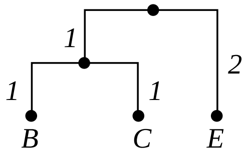
We want to reconstruct an ultrametric tree using UPGMA. As $d_{BC} = 2$ is the smallest distance we join B and C into a new cluster (BC) with depth 1:



We compute a new distance matrix. E.g. $d_{A(BC)} = (1 \cdot 8 + 1 \cdot 8)/(1 + 1) = 8$ etc.

	A	(BC)	D	E
A	0	8	12	8
(BC)		0	12	4
D			0	12
E				0

We join (BC) and E with depth 2:

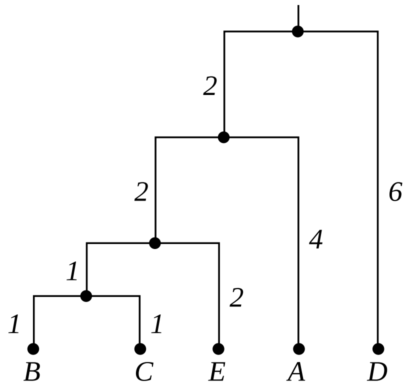


We obtain distances to $((BC)E)$, e.g. $d_{A((BC)E)} = (1 \cdot 8 + 3 \cdot 8)/(1 + 3) = 8$. Note that single linkage, complete linkage and WPGMA would give the same result. This is due to distances being ultrametric.

The modified distances are

	A	$((BC)E)$	D
A	0	8	12
$((BC)E)$		0	12
D			0

We join $((BC)E)$ and A with depth 4 and finally $(((BC)E)A)$ and D are left to join:



UPGMA was originally developed for phenetics [17], i.e. for constructing phenograms reflecting phenotypic similarities rather than evolutionary distances. Given an approximately constant rate of evolution, that is if observed distances are close to an ultrametric, it is also suited for phylogeny reconstruction, and it is the most commonly used clustering method for this purpose.

While UPGMA assigns equal weight to each original distance, WPGMA does not, therefore it is called weighted. Single linkage and complete linkage are somewhat extreme cases of clustering. While complete linkage clusters are usually very compact (each element in a cluster is connected to each other element), single linkage clusters may contain pairs of elements that by direct comparison are rather dissimilar when there is a “path” of connecting elements between them.

All mentioned clustering methods give the same result when the data are ultrametric, but they can differ when they are not.

6.3 Additive Trees

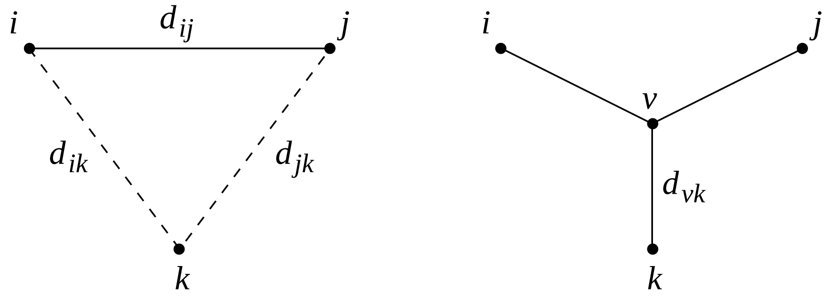
We have seen that ultrametric trees are rooted trees and imply the existence of a molecular clock. But rates of evolution vary among species, among gene families, among sites in molecular sequences and generally in the course of sequence evolution. *Additive trees* do not presume a constant evolutionary rate nor do they make any assumption about the rooting and therefore reflect our ignorance as to where the common ancestor lies. Given an additive distance matrix there is exactly one tree topology that allows for realization of an additive tree. We will show how to reconstruct it in Section 6.3.1.

If one wants to construct a tree T from a distance matrix d^M, then the aim is that distances d^T are as similar as possible to the observed distances d^M. In Sections 6.3.2 and 6.3.3 we discuss two methods that optimize a simple objective function when observed distances d^M are not additive, the Fitch-Margoliash algorithm and the Minimum Evolution method, which aim at reconstructing an additive tree T with distances d^T being as similar as possible to observed distances d^M. Finally in Section 6.3.5 we present the popular and heuristic method called Neighbor Joining.

6.3.1 Exact Reconstruction of Additive Trees

An additive metric can be represented as a unique additive tree which can be reconstructed in time complexity $O(n^2)$ [27]. The algorithm successively inserts objects into intermediate trees until no objects are left to insert.

We use the following rationale: Given an intermediate tree T' containing leaf i and leaf j, we test if we can insert an edge connecting leaf k to the intermediate tree along the path connecting i and j. We denote the node connecting i, j and k as v and the weight of the edge being inserted as d_{vk}.



We observe that

$$d_{ik} + d_{jk} = d_{iv} + d_{vk} + d_{jv} + d_{vk} = 2 \cdot d_{vk} + d_{ij}$$

and therefore the weight of the inserted edge would be

$$d_{vk} = \frac{1}{2}(d_{ik} + d_{jk} - d_{ij})$$

and respectively

$$\begin{aligned} d_{iv} &= d_{ik} - d_{vk} \\ d_{jv} &= d_{jk} - d_{vk} \end{aligned}$$

Given a set of objects O and an additive metric d on O, the algorithm first picks two arbitrary objects $i, j \in O$ and connects them by an edge with weight d_{ij}. This gives the first intermediate tree T'. Then, iteratively each object $k \in O$ not yet in T' is connected to T' by an edge e^k by the following algorithm.

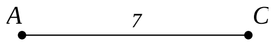
1. Pick a pair of leaves $i, j \in T'$.
2. Compute the weight of e^k by means of the above rationale.
3. If the insertion of e^k in T' implies that e^k has to be inserted inside an edge, split that edge, insert a node and attach e^k to that node; otherwise (if the insertion point is a node), replace j (or i) by a leaf from the subtree along the edge at this node not leading towards i or j and continue with 2.

To see that the algorithm runs in $O(n^2)$ time, observe that there are $n-2$ iterations, each requiring a linear number of weight computations. The directed traversal of the tree in order to test if the branching point is a node or not can be done in time proportional to the number of nodes traversed with a few data simple structures.

Example: Given a set of objects $O = \{A, B, C, D, E\}$ and an additive metric d on O

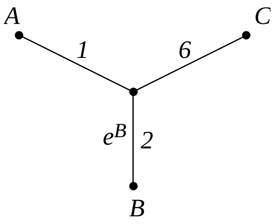
	A	B	C	D	E
A	0	3	7	10	7
B		0	8	11	8
C			0	9	6
D				0	5
E					0

we first pick two arbitrary objects, say A and C, and connect them by an edge of weight $d_{AC} = 7$ to set up the first intermediate tree T':



We connect B by an edge e^B to T'. The weight of e^B is

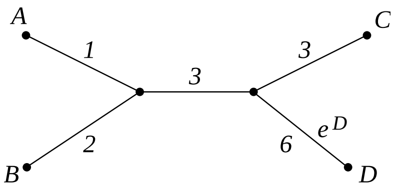
$$\frac{d_{AB} + d_{CB} - d_{AC}}{2} = \frac{3+8-7}{2} = 2$$



We try to connect D by an edge e^D branching off the path between B and C. The weight of e^D would be

$$\frac{d_{BD} + d_{CD} - d_{BC}}{2} = \frac{11+9-8}{2} = 6$$

and inserting e^D on the edge branching off to C is therefore consistently possible:

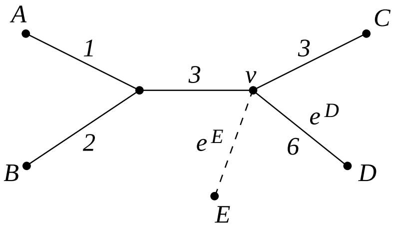


Finally we have to connect E by an edge e^E. Try to let e^E branch off the path between B and C. The weight of e^E would be

$$d_{vE} = \frac{d_{BE} + d_{CE} - d_{BC}}{2} = \frac{8+6-8}{2} = 3$$

and hence

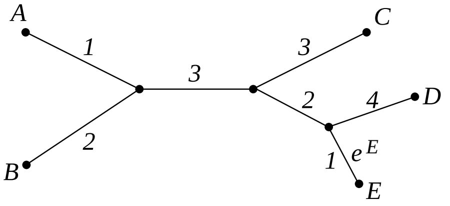
$$d_{Bv} = d_{BE} - d_{vE} = 8 - 3 = 5.$$



This implies that e^E has to be inserted at node v, and hence the procedure is repeated with C being replaced by D. Choosing the path between B and D, the weight of e^E is

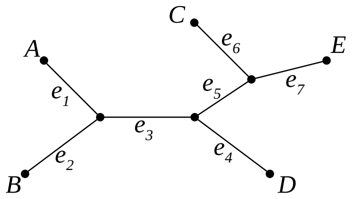
$$\frac{d_{BE} + d_{DE} - d_{BD}}{2} = \frac{8+5-11}{2} = 1$$

and as $d_{DE} = 5$, e^E branches off e^D:



6.3.2 Least Squares (Fitch-Margoliash)

In practice, a distance measure d^M on a set of homologous sequences hardly will form an additive metric. However, if d^M is close to being additive, it is straight forward to fit d^M to additive distances d^T of a tree. Fitch-Margoliash and other methods are based on a familiy of objective functions called *least squares*.



Consider the above tree T with its set of leaves $O = \{A, B, C, D, E\}$ and weighted edges $e_1, e_2, ..., e_7$. In the following table we assign 1 to a pair of leaves (X, Y) and an edge e_j whenever e_j belongs to the simple path between X and Y, and 0 otherwise.

	e_1	e_2	e_3	e_4	e_5	e_6	e_7
(A,B)	1	1	0	0	0	0	0
(A,C)	1	0	1	0	1	1	0
(A,D)	1	0	1	1	0	0	0
(A,E)	1	0	1	0	1	0	1
(B,C)	0	1	1	0	1	1	0
(B,D)	0	1	1	1	0	0	0
(B,E)	0	1	1	0	1	0	1
(C,D)	0	0	0	1	1	1	0
(C,E)	0	0	0	0	0	1	1
(D,E)	0	0	0	1	1	0	1

This table, interpreted as a $(\frac{n(n-1)}{2} \times 2n-3)$ matrix M^T, is called the *path edge incidence matrix.*

Further, let $\vec{e}$ be the vector of the $2n-3$ edge weights e_i. We see that

$$\vec{d^T} = M^T \vec{e}$$

is a vector holding the tree distances between leaves of T.

Fitch and Margoliash [11] define the disagreement between a tree and the distance measure by

$$E := ||\vec{d^T} - \vec{d^M}||^2 = \sum_{i<j} |d^T_{ij} - d^M_{ij}|^2.$$

One then wants to find a tree topology and edge lengths such that E is minimal. (To be precise, Fitch and Margoliash weight the difference of d^T and d^M by their measured distances, i.e. $E :=$

$\sum_{i<j} |d^T_{ij} - d^M_{ij}|^2/(d^M_{ij})^2$, because they want to minimize the square of the *relative error*, i.e. they assume that the uncertainty of the measurement is by the same *percentage* for all measurements.)

This can be solved by linear algebra: For a given tree topology (represented by a matrix M^T), one can find the edge lengths $\vec{e}$ that minimize

$$E = ||\vec{d}^T - \vec{d}^M||^2 = ||M^T\vec{e} - \vec{d}^M||^2,$$

e.g. by solving

$$M^T\vec{e} = \vec{d}^M$$

using singular value decomposition. A numerical solution is also possible and often faster.

This allows to optimize the branch lengths for a given topology. Still, one has to repeat this test for all (or many) topologies. The least squares tree is then the tree that minimizes E.

In practice this exact method takes very long. That is why Fitch and Margoliash suggest a heuristic clustering algorithm to find tree topologies that have small least squares error.

Fitch-Margoliash Heuristic. It is clear that for three taxa A, B, C always a correct tree can be computed: For the internal node M the following must hold: $d_{AM} = (d_{AB} + d_{AC} - d_{BC})/2$ (see Section 6.3.1). Similarly for sets of taxa, where one can use the average distances of all pairs of taxa from both sets.

1. Initially each taxon forms its own set.
2. While there is more than one set, do:
    - (i) Join the pair of sets (i, j), such that d_{ij} is minimal.
    - (ii) Let x be the set consisting of all taxa not in i or j.
    - (iii) Compute the exact edge lengths for the tree $((i, j), x)$ as described above.
    - (iii) Recompute the distances as arithmetic averages over all possible pairs of proteins from the two groups: $d_{u=i\cup j,k}$ for each k like in UPGMA.
3. Finally fit the branch lengths by minimizing the least squares error E as described above.

The whole procedure can be repeated several times with other choices when there are ties, and finally the tree with the smallest error is taken.

6.3.3 Minimum Evolution

Alternatively, Waterman *et al.* [27] have formulated a linear program. The objective function of the linear program is in spirit similar to the Steiner tree setup: the overall length of the tree is minimized. The idea is to use the least squares criterion to fit the branch lengths, but to evaluate and compare trees, one uses the “tree length”

$$L := \sum_{i=1}^{2n-3} |e_i|$$

where the e_i are the $2n-3$ edge lengths, computed from the pairwise distances between the sequences as above. The tree minimizing L is called the *minimum evolution tree* (ME tree). There are two constraints to the linear program:

(i) all the branch lengths are non-negative

$$e_i \geq 0$$

(ii) the alignment distance between two sequences may underestimate the number of changes that have occurred between the two sequences in the course of evolution. Therefore, for any pair of sequences, the tree distance is not allowed to be smaller than the measured distance:

$$d_{ij}^T \geq d_{ij}^M \qquad \forall i, j.$$

Simulations have shown that ME consistently outperforms Least Squares.

6.3.4 Fast Minimum Evolution

The following is from [7].

"Normal" Minimum Evolution (Rzhetsky/Nei) in practice:
1. Apply neighbor joining to get an initial tree. $O(n^3)$
2. Refinement by branch swapping/nearest neighbor interchange, that is accepted whenever the amount of evolution decreases. $O(pn^3)$ where p is the number of (tried) swaps

Fast Minimum Evolution (Desper/Gascuel):
1. Greedy addition algorithm. $O(n^2)$
2. FASTNNI. $O(np)$ where p is the number of swaps (this is based on a clever pre-processing method)

6.3.5 Neighbor Joining

The *neighbor joining* (NJ) method is similar to cluster analysis in some ways. The individual taxa are iteratively grouped together, forming larger and larger clusters of taxa. In contrast to UPGMA, neighbor joining does not assume a molecular clock, but it assumes that observed distances are close to an additive metric. Given an additive metric, the neighbor joining method identifies the correct tree [24] and it also correctly reconstructs trees if additivity only holds approximately [1].

As neighbor relationships of nodes in a binary uniquely define the tree topology, successively identifying neighbors is a way to reconstrut the tree. In each iteration of the NJ algorithm, every pair of taxa is evaluated for being neighbors, and if so, they are grouped together to form a new taxon for the next iteration. Here, the notion of neighborhood is defined as follows:

Definition Two taxa are *neighbors* in a tree if the path between them contains only one node.

Note that neighbors do not need to have the smallest distance in the distance matrix. For an example, see the tree in Figure 6.4.

The corresponding distance matrix d is:

	A	B	C	D
A	0	3	4	5
B		0	5	4
C			0	7
D				0

UPGMA would pick taxa A and B to cluster them together, since d_{AB} is the smallest distance. Actually A and B are not neighbors, but A and C are. This effect is due to the long edge with weight 3 (corresponding to a high rate at which mutations have accumulated) branching off to C.

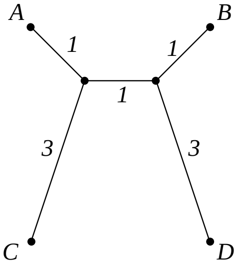


Figure 6.4: Neighbors not necessarily have the smallest distance

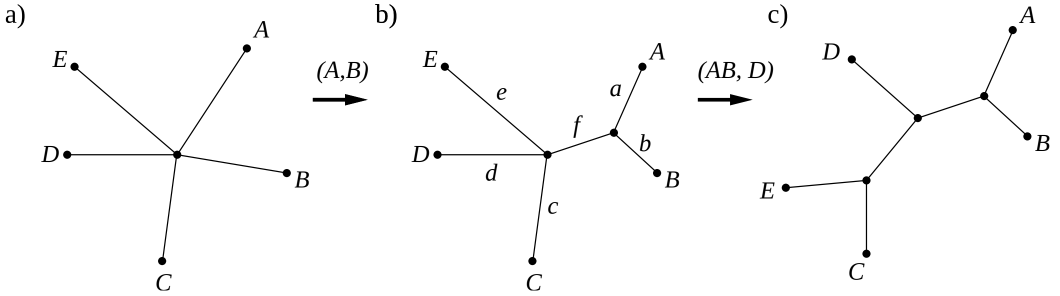


Figure 6.5: Neighbor Joining successively decomposes a star tree by identifying neighbors. Pairs of neighbors are written in parenthesis.

After presenting the central theorem and the algorithm, we show that Neighbor Joining correctly identifies the neighbor relationsships for the tree in Figure 6.4

The concept to identify neighbors is a variation of the Minimum Evolution principle (see also 6.3.3): A star tree is decomposed such that the tree length is minimized in each step. Consider the star tree with N leaves in Figure 6.5 a). The star tree corresponds to the assumption that there is no clustering of taxa. In general there is a clustering of taxa and if so, the overall tree length (the sum of all branch lengths) S_F of the true tree or the final NJ tree (see Figure 6.5 c)) is smaller than the overall tree length of the star tree S_0. Consider the tree in Figure 6.5 b) with resolved neighbors A and B. It is clear that the tree length S_{AB} of this tree is smaller than S_0. A general formula for the tree length S_{ij} of a tree like in Figure 6.5 b) when considering taxa i and j as neighbors is

$$S_{ij} = \sum_{\substack{k=1 \\ k \neq i,j}}^{N} \frac{d_{ki} + d_{kj}}{2(N-2)} \quad + \quad \frac{d_{ij}}{2} \quad + \quad \sum_{\substack{k<l \\ k,l \neq i,j}}^{N} \frac{d_{kl}}{N-2}$$

where N is the number of taxa. Computation of S_{AB} yields

$$S_{AB} = (3a + 3b + 6f + 2c + 2d + 2e) \cdot \frac{1}{6} \; + \; \frac{a+b}{2} \; + \; (2c + 2d + 2e) \cdot \frac{1}{3} = a + b + f + c + d + e$$

Theorem: Given an additive tree T. O is the set of leaves of T. Values of S_{ij} are computed by means of the path metric d^T. Then $m, n \in O$ are neighbors in T, if $S_{mn} \leq S_{ij} \quad \forall \, i, j \in O$.

A simple proof makes use of the four point condition (see section 6.1). It enables us to identify a pair of neighbors given additive distances between a set of taxa by computing S_{ij} for all pairs of taxa and choosing taxa i and j showing the smallest S_{ij} value. The identified neighbors are combined into one composite taxon and the procedure is repeated. We rewrite S_{ij}:

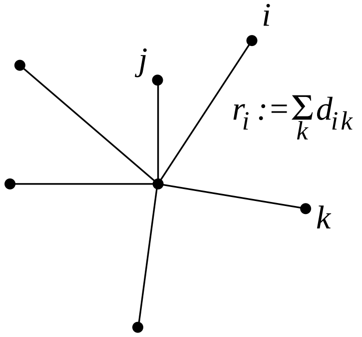
$$\begin{aligned} S_{ij} &= \frac{1}{2(N-2)}\Big(2\cdot\sum_{\substack{k<l\\k,l\neq i,j}}^{N} d_{kl} + \sum_{\substack{k=1\\k\neq i,j}}^{N}(d_{ki}+d_{kj})\Big) + \frac{d_{ij}}{2} \\ &= \frac{1}{2(N-2)}\Big(2\cdot\sum_{i<j}^{N} d_{ij} - r_i - r_j\Big) + \frac{d_{ij}}{2} \end{aligned}$$

with $r_i := \sum_{k=1}^{N} d_{ik}$. Since the sum $\sum_{i<j}^{N} d_{ij}$ is the same for all pairs of i and j, we can replace S_{ij} by

$$M_{ij} := d_{ij} - \frac{r_i + r_j}{N-2}$$

for the purpose of easier computation of relative values of S_{ij}.

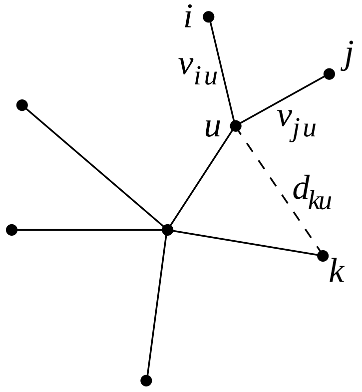
Algorithm: Given distances d_{ij} between members of a set O of N objects. Represent the objects as terminal nodes in a starlike tree:



1. For each terminal node i compute
$$r_i := \sum_{k=1}^{N} d_{ik}.$$
2. For all pairs of terminal nodes (i,j) compute
$$M_{ij} := d_{ij} - \frac{r_i + r_j}{n-2}.$$
Let (i,j) be a pair with minimal value M_{ij} for $i \neq j$.
3. Join nodes i and j into a new terminal node u. The branch lengths from u to i and j are:
$$v_{iu} = \frac{d_{ij}}{2} + \frac{r_i - r_j}{2N-4} \quad \text{and} \quad v_{ju} = d_{ij} - v_{iu}.$$

4. Obtain the distances from u to another terminal node k by

$$d_{ku} = \frac{d_{ik} + d_{jk} - d_{ij}}{2}.$$



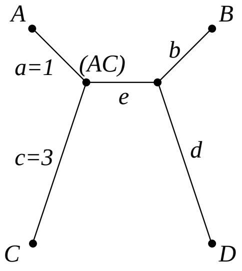
5. Delete i and j from the set of objects. If there are more than two clusters left, continue with Step 1

Now we will give an example for a tree reconstruction given an (exact) adddditive metric by means of the NJ algorithm.

Example The path metric d^T for the tree in Figure 6.4 is given by the following distances:

	A	B	C	D
A	0	3	4	5
B		0	5	4
C			0	7
D				0

We reconstruct the tree by means of the NJ algorithm. In the first iteration A, B, C and D are the terminal nodes of the star tree and we compute $r_A = r_B = 12$, $r_C = r_D = 16$ and $M_{AB} = d_{AB} - (r_A + r_B)/N - 2) = 3 - 24/2 = -9$, $M_{AC} = M_{BD} = 4 - 28/2 = -10$, $M_{AD} = M_{BC} = 5 - 28/2 = -9$, $M_{CD} = 7 - 32/2 = -9$. M_{AC} and M_{BD} have the smallest value, that is, the NJ algoritm correctly identifies A and C as well as B and D as neighbors. We combine A and C into a composite taxon (AC).



The edge lengths a and c are $a = d_{AC}/2 + (r_A - r_c)/(2N-4) = 2 + (-4/4) = 1$ and $c = d_{AC} - a = 4 - 1 = 3$. New distances of the composite taxon (AC) to B and D are $d_{(AC)B} = (d_{AB} + d_{CB} - d_{AC})/2 =$

$(3+5-4)/2 = 2$ and $d_{(AC)D} = 4$. We delete A and C from the set of objects and do the second iteration with the distance matrix

	(AC)	B	D
(AC)	0	2	4
B		0	4
D			0

There are three terminal nodes left and therefore we expect them all to be pairwise neighbors. Computations yield $r_{(AC)} = 6$, $r_B = 6$, $r_D = 8$ and $M_{(AC)B} = d_{(AC)B} - (r_{(AC)} + r_B)/(N-2) = 2 - 12 = -10$, $M_{(AC)D} = 4 - 14 = -10$, $M_{(BD)} = -10$. Grouping (AC) and B together into (ACB) we obtain $e = d_{(AC)B}/2 + (r_{AC} - r_B)/2 = 1$ and $b = 2 - 1 = 1$. Now there's only one distance left to compute: $d = d_{(ACB)D} = (d_{(AC)D} + d_{BD} - d_{(AC)B})/2 = (4+4-2)/2 = 3$. The NJ tree is the same as the true tree (Figure 6.4).

Chapter 7

Split Decomposition

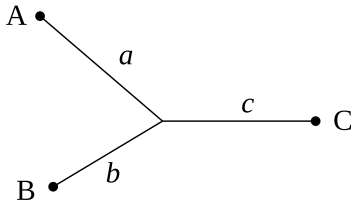
7.1 Introduction

The methods for reconstructing (ultrametric or additive) trees we have looked at before will always construct a *tree* even if the underlying distance data is not at all tree-like. In such a case a tree-like relationship will be suggested which is not present in the data. Unfortunately in most cases the methods do not even tell how close the approximation of the data to the resulting tree is. And even if they do (as in the least-squares method), only a single quantity is returned without any information as to specific regions where the dissimilarity occurs or what alternative trees might be.

A method called *split decomposition* developed by Bandelt and Dress [2, 3] allows both to quantify the tree-likeness of given distance data and present alternative relationships. A measured dissimilarity matrix d^M is decomposed into a number of *splits* (binary partitions of the set of taxa) weighted by *isolation indices* (indicating the strength of the split), plus a residual *noise* term. This is motivated by the assumption that measured distance data may underly a systematic error. Assume that $d^M = d^T + d^E$ where r^T is the true tree-like relationship and d^E is an error term. If d^E itself represents a tree (different from the true one), then the two sets of splits will overlay. Ab initio it will not be possible to distinguish which splits are the true ones, and which result from the error term, but if the true splits are stronger than the error splits, one can assume that those splits with the larger isolation index belong to d^T rather than to d^E.

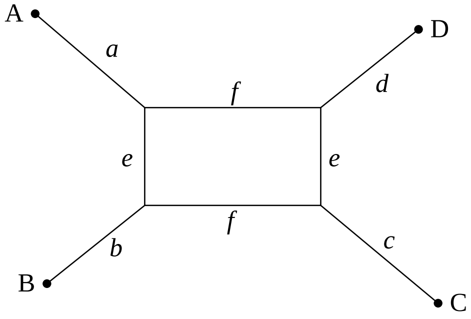
7.2 Basic Idea

First remember the case of three taxa A, B, C and three distances d_{AB}, d_{AC}, d_{BC}. As long as the triangle inequality ($d_{AC} \leq d_{AB} + d_{BC}$) holds, it is always possible to compute a unique tree: We have three conditions and three unknowns (the lengths of the three terminal edges) a, b, c:



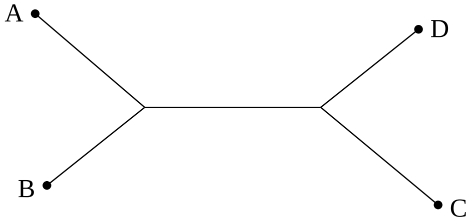
$$\begin{aligned} a &= \frac{1}{2}(d_{AB} + d_{AC} - d_{BC}) \\ b &= \frac{1}{2}(d_{AB} + d_{BC} - d_{AC}) \\ c &= \frac{1}{2}(d_{AC} + d_{BC} - d_{AB}) \end{aligned}$$

In the case of four taxa, a tree is no longer uniquely defined: we have six conditions, but a tree with four leaves has only five edges. However, the following diagram, which shown a generalized "tree" whose internal edges is replaced by a rectangle, has six unknown edge lengths a, b, c, d, e, f:



$$\begin{aligned} a &= \frac{1}{2}(d_{AB} + d_{AD} - d_{BD} \\ b &= \frac{1}{2}(d_{AB} + d_{BC} - d_{AC} \\ c &= \frac{1}{2}(d_{BC} + d_{CD} - d_{BD} \\ d &= \frac{1}{2}(d_{AD} + d_{CD} - d_{AC} \\ e &= \frac{1}{2}(d_{AC} + d_{BD} - d_{AD} - d_{BC}) \\ f &= \frac{1}{2}(d_{AC} + d_{BD} - d_{AB} - d_{DC}) \end{aligned}$$

Now assume that the phylogenetic relationship is such that the true split separates the two pairs (A,B) and (C,D).



Then due to the four-point condition the true tree distances d^T are related as follows:

$$d^T_{AB} + d^T_{CD} < d^T_{AC} + d^T_{BD} = d^T_{AD} + d^T_{BC}.$$

As discussed above, the *measured* (evolutionary) distances d^M will usually not fulfill this relationship, normally not even both orderings

$$d^M_{AB} + d^M_{CD} < d^M_{AC} + d^M_{BD} \quad \text{and} \quad d^M_{AB} + d^M_{CD} < d^M_{AC} + d^M_{BD}.$$

However, one could hope that at least $d^M_{AB} + d^M_{CD}$ is not the largest of the three sums.

7.3 Definitions

Based on this idea, one can define a *split with respect to* d^M or a d^M*-split* (or d-split when it is clear from the context that we mean d^M) between a group J and a group K of taxa as follows: For any two taxa i, j from J and k, l from K, the sum $d_{ij} + d_{kl}$ must not be the largest among the three possible sums, i.e.

$$d_{ij} + d_{kl} < \max\{d_{ik} + d_{jl}, d_{il} + d_{jk}\}.$$

Obviously, for n taxa there can be at most $\binom{n}{2}$ such splits. This number, however, is considerably larger than $2n - 3$, the maximal number of splits (edges) in an additive tree, hence splits define a relationship more general than a tree. Typically the number of d-splits observed in real data is only about $2n$, though.

Each d-split receives a positive weight, its *isolation index* defined as follows:

$$\alpha_{J,K} = \frac{1}{2} \min_{\substack{i,j \in J \\ k,l \in K}} (\max\{d_{ij} + d_{kl}, d_{ik} + d_{jl}, d_{il} + d_{jk}\} - d_{ij} - d_{kl}).$$

Note that all partitions that do not qualify as d-splits have isolation index 0. Further, the isolation index of a split in an additive tree is the length of the edge that defines the split.

The *split metric* $\delta_{J,K}$ defined by a split J, K assigns distance 0 to two taxa that are both in J or both in K, and 1 otherwise. Further, define d^1 to be the sum of all split metrics, weighted by their isolation indices:

$$d^1 = \sum_{d-\text{splits } J,K} \alpha_{J,K} \delta_{J,K}.$$

Then it can be shown [2] that d can be approximated as follows: $d = d^0 + d^1$ where d^0 is a metric that does not contain any further splits. The proportion of d that can be split is called the *splittable percentage*:

$$\rho := \left(\sum_{\text{taxa } i,j} d^1_{ij} / \sum_{\text{taxa } i,j} d_{ij} \right) \cdot 100\%.$$

7.4 Computation of the d-Splits

The set of d-splits of a distance matrix of n taxa $1, 2, \ldots, n$ can be computed by the following recursive algorithm:

1. Start with taxa $1, 2, 3, 4$. This case can easily be solved exhaustively.
2. For $i = 5, \ldots, n$ do the folowing:
   (i) Assume the d-splits restricted to the subset $\{1, \ldots, i-1\}$ given. For each d-split J, K of this subset, test if $J \cup \{i\}, K$ or $J, K \cup \{i\}$ qualifies as a d-split.
   (ii) Test if $\{1, \ldots, i-1\}, \{i\}$ is a d-split.

The algorithm can easily be implemented in $O(n^6)$ time.

Graph drawing. For a tree this is trivial: splits correspond to edges. In general, the resulting graphs are subgraphs of the $\binom{n}{2}$-dimensional hypercube, but usually they are not too weird, often planar. A split corresponds to several parallel edges, the length of which is proportional to the split's isolation index.

The splits diagram can be contructed incrementally. The outcome is not unique; it depends on the order in which the splits are processed.

Tree selection. If one wants, one can choose a set of splits that define a tree by greedily selecting those splits with maximal isolation index if they are compatible (every pair has parts with empty intersection) with previously selected ones.

Program. The splits method is implemented in a program called *Splitstree* that can be used online at

`http://bibiserv.techfak.uni-bielefeld.de/splits`

or downloaded from

`http://www-ab.informatik.uni-tuebingen.de/software/splits/welcome_en.html`

The program

- finds the splits,
- draws a picture of the splits diagram,
- and calculates the splittable percentage.

Case studies. The paper [3] by Bandelt and Dress contains a number of interesting case studies that illustrate the splits method.

7.5 NeighborNet

The following is from [4].

Combination of Neighbor Joining and SplitsTree

- Incremental tree construction similar to NJ.
- Result is a network as in SplitsTree, but with a better resolution
- $O(n^3)$ running time

Part IV

Likelihood Methods

Chapter 8

Modeling Sequence Evolution

8.1 Basics on Probability

8.1.1 Events and Probabilities

Throwing a die may be seen as an experiment: We do not know in advance which number will turn up. All we know is, that a number between 1 and 6 will turn up.

Definition. The set of all possible outcomes of an experiment is called *sample space* Ω. A subset $A \subset \Omega$ is called *event*. Elements $\omega \in \Omega$ are called *elementary events*. The event Ω is called the *certain event* and the event $\emptyset$ is called the *null event*. The *complement* of an event A is denoted by A^C. Events A and B are called *disjoint* if $A \cap B = \emptyset$.

Example. A die is thrown. The sample space is $\Omega = \{1, 2, 3, 4, 5, 6\}$. The event $A = \{2, 4, 6\}$ means, the outcome of the experiment is an even number

We want to assign real numbers representing probabilities to subsets of Ω, that is to events. If Ω is infinite, e.g. $\Omega = \mathbb{R}$, it is not possible to reasonably assign probabilities to all members of the *power set* denoted by $\{0, 1\}^\Omega$ which contains all subsets of Ω. Therefore we introduce the collection $\mathcal{F}$ of subsets of Ω containing all events of interest.

Definition. A collection $\mathcal{F}$ of subsets of Ω is called a σ*-field* if it satisfies the following conditions:

(i) $\emptyset \in \mathcal{F}$

(ii) if $A_1, A_2, ... \in \mathcal{F}$ then $\bigcup_{i=1}^{\infty} A_i \in \mathcal{F}$

(iii) if $A \in \mathcal{F}$ then $A^C \in \mathcal{F}$

Example. A die is thrown. The sample space is $\Omega = \{1, 2, 3, 4, 5, 6\}$. $\mathcal{F} = \{\emptyset, \{2, 4, 6\}, \{1, 3, 5\}, \Omega\}$ is a σ-field.

We proceed by defining the properties of a function Pr assigning probabilities to events.

Definition. A *probability measure* Pr on $(\Omega, \mathcal{F})$ is a function $Pr : \mathcal{F} \rightarrow [0,1]$ satisfying

(i) $Pr(\emptyset) = 0, \qquad Pr(\Omega) = 1;$

(ii) if $A_1, A_2, ...$ is a collection of disjoint members of $\mathcal{F}$ then

$$Pr\Big(\bigcup_{i=1}^{\infty} A_i\Big) = \sum_{i=1}^{\infty} Pr(A_i)$$

The triple $(\Omega, \mathcal{F}, Pr)$ is called a *probability space*, $Pr(A)$ is called the *probability* of event A.

Example. A die is thrown. The sample space is $\Omega = \{1,2,3,4,5,6\}$ and we can take $\mathcal{F} = \{0,1\}^{\Omega}$. ω_i denotes the elementary event that number i turns up. Suppose the die is fair, that is, each elementary event ω_i has the same chance to occur. As $Pr(\Omega) = 1$ and $\bigcup_{i=1}^{6} \omega_i = \Omega$ where $\omega_i \cap \omega_j = \emptyset \; \forall \; i, j, \; i \neq j$, we see that $Pr(\omega_i) = p = 1/6$. The probability for the event $A = \{\omega_1, \omega_2\}$ that '1' or '2' turns up is $Pr(A) = 2p = 1/3$.

8.1.2 Conditional probability

Suppose, we have some prior knowledge of the outcome of an experiment.

Definition. The *conditional probability* of an event A given that another event B occurs is

$$Pr(A|B) = \frac{Pr(A \cap B)}{Pr(B)}, \quad \text{if } Pr(B) > 0$$

Example. A fair die is thrown. What is the probability for the event A that the number turning up is bigger than 1 given we know that the number is even? The probability for the event B that the number is even is $Pr(B) = 1/2$. We see that $Pr(A \cap B) = Pr(\{2,3,4,5,6\} \cap \{2,4,6\}) = Pr(\{2,4,6\}) = 1/2$. Therefore we get $Pr(A|B) = (1/2)/(1/2) = 1$.

Lemma. If A and B are events and $Pr(B) > 0$ and $Pr(B^C) > 0$ then

$$Pr(A) = Pr(A|B) \; Pr(B) \; + Pr(A|B^C) \; Pr(B^C)$$

Proof. $A = (A \cap B) \cup (A \cap B^C)$. As $(A \cap B) \cap (A \cap B^C) = \emptyset$ we get $Pr(A) = Pr(A \cap B) + Pr(A \cap B^C) = Pr(A|B)Pr(B) + Pr(A|B^C)Pr(B^C)$. $\square$

Example. (to be done)

8.1.3 Bayes's formula

From the definition of conditional probability we obtain

$$Pr(A \cap B) = Pr(A|B)\ Pr(B) = Pr(B \cap A) = Pr(B|A)\ Pr(A)$$

Solving for $Pr(A|B)$ we obtain *Bayes's formula*

$$Pr(A|B) = \frac{Pr(B|A)\ Pr(A)}{Pr(B)}$$

which often turns out to be useful in computation of conditional probabilities.

Example. (to be done)

8.1.4 Independence

Definition. Two events A and B are *independent* if

$$Pr(A \cap B) = Pr(A)Pr(B).$$

Example. A fair die is thrown. Suppose event A is that the number turning up is bigger than 2 and event B is that the number turning up is even. $A \cap B = \{4, 6\}$ and therefore $Pr(A \cap B) = 1/3$. We see that $Pr(A)Pr(B) = 2/3 \cdot 1/2 = \frac{1}{3} = Pr(A \cap B)$. Events A and B are independent.

Example. Two fair dice are rolled. The sample space is $\Omega = \{(a, b) : a = 1, ..., 6;\ b = 1, ..., 6\}$ and $|\Omega| = 36$. Let events A and B be that a '1' turns up on the first and the second die, respectively. The probability for the event $(1, 1)$ that two times '1' turns up is $Pr(A \cap B) = 1/6 \cdot 1/6 = 1/36$. A and B are independent.

8.1.5 Random variables

Often we are not directly interested in the outcome of an experiment but in a function on the outcome, e.g. in the sum of numbers when rolling two dice.

Definition. A *random variable* is a function $X : \Omega \to \mathcal{X}$. $\mathcal{X}$ may be any set.

Example. Two fair dice are rolled. The sample space is $\Omega = \{(a, b) : a = 1, ..., 6;\ b = 1, ..., 6\}$. We define the random variable X by $X(\omega) := a + b$ for $\omega = (a, b) \in \Omega$. Therefore $\mathcal{X} = \{2, 3, ..., 12\}$. The probability that X takes value '3' is $Pr(X = 3) = Pr(\{(1, 2)\}) + Pr(\{(2, 1)\}) = 1/18$.

8.2 Markov Chains

Assuming that sites in DNA and amino acid sequences evolve independently from each other, evolution of molecular sequences is commonly modeled by means of a Markov chain at each site.

8.2.1 Time Discrete Markov Chains

Definition. A *time discrete Markov chain* is a sequence of random variables $X_n,\ n \in \mathbb{N}_0$ taking values of a finite *set of states* $\mathcal{A}$ where

(i) X_0 is sampled according to an *initial distribution* $\pi^{(0)}$:

$$\pi_i^{(0)} = Pr(X_0 = i), \qquad i \in \mathcal{A}$$

(ii) the *Markov property* has to be satisfied:

$$Pr(X_n = x_n | X_{n-1} = x_{n-1}, ..., X_0 = x_0) = Pr(X_n = x_n | X_{n-1} = x_{n-1}),$$

for any $n \in \mathbb{N}$ and any states $x_0, ..., x_n \in \mathcal{A}$.

When modeling sequence evolution, $\mathcal{A}$ will be the set of amino acid residues $\mathcal{A} = \{1, 2, ..., 20\}$ or the set of nucleotides $\mathcal{A} = \{1, 2, 3, 4\}$ (e.g. $1 = A$, $2 = G$, $3 = C$, $4 = T$), respectively. Thus we can think of the Markov Chain as describing the behavior of a site in a molecular sequence in time. The Markov property means that the conditional probability for the observation of a state at a given time point only depends on the previous time point. The conditional probability for a state to reach another state is called *transition probability.*

We assume that the Markov chain is *time homogeneous*, that is for each pair of states $i, j \in \mathcal{A}$ the transition probability $Pr(X_n = j | X_{n-1} = i)$ is the same for all $n \in \mathbb{N}$.

Transition probabilities are held in the *transition probability matrix* P with entries

$$P_{ij} = Pr(X_n = j | X_{n-1} = i), \quad i, j \in \mathcal{A}$$

P is a stochastic matrix, i.e. each entry is nonnegative, $P_{ij} \geq 0\ \forall i, j$, and each row sums up to 1, $\sum_j P_{ij} = 1$. The pair $(\pi^{(0)}, P)$ specifies a unique homogeneous Markov chain.

We want to compute transition probabilities for k steps of the Markov chain. Let's start with two steps and $\mathcal{A} = \{1, 2, 3, 4\}$. We obtain

$$\begin{aligned} Pr(X_{n+1} = 4 | X_{n-1} = 1) \ = \ & \quad Pr(X_n = 1 | X_{n-1} = 1)\ Pr(X_{n+1} = 4 | X_n = 1) \\ & + Pr(X_n = 2 | X_{n-1} = 1)\ Pr(X_{n+1} = 4 | X_n = 2) \\ & + Pr(X_n = 3 | X_{n-1} = 1)\ Pr(X_{n+1} = 4 | X_n = 3) \\ & + Pr(X_n = 4 | X_{n-1} = 1)\ Pr(X_{n+1} = 4 | X_n = 4) \\ = \ & \sum_{k \in \mathcal{A}} P_{1k} P_{k4} \end{aligned}$$

We denote $P^{(1)} = P^1 = P$ as the 1-step transition matrix. We see, that the 2-step transition matrix is $P^{(2)} = P^2 = P \cdot P$ and the k-step transition matrix is $P^{(k)} = P^k = PP^{k-1}$.

Definition. A Markov chain is *irreducible* if for any two states $i, j \in \mathcal{A}$ there exists $k \in \mathbb{N}$ such that $P_{ij}^{(k)} > 0$.

A Markov chain is irreducible if it is possible for the chain to reach each state from each state. This generally holds for the Markov chains in our applications.

$\pi^{(k)}$ is a row vector holding the distribution of the states after k steps. Given the initial distribution $\pi^{(0)}$ we obtain

$$\pi_j^{(k)} = \sum_{j \in \mathcal{A}} \pi_j^{(0)} \cdot P_{ij}^{(k)},$$

$$\pi(k) = \pi^{(0)} \cdot P^{(k)}.$$

Definition. A distribution π is a *stationary distribution* on $\mathcal{A}$ if

$$\pi_j = \sum_{i \in \mathcal{A}} \pi_i P_{ij} \qquad j \in \mathcal{A},$$

$$\pi = \pi P.$$

If $\pi^{(0)} = \pi$, every X_n is distributed as π. Furthermore each irreducible homogeneous Markov chain converges against its stationary distribution:

Theorem. Given an irreducible homogeneous Markov Chain $(\pi^{(0)}, P)$, there exists exactly one stationary distribution π and

$$\lim_{k \to \infty} P_{ij}^{(k)} = \pi_j \qquad \forall\ i.$$

8.2.2 Time continuous Markov Chains

In this subsection we will transfer the notions from time discrete Markov chains to time continuous Markov chains.

Definition. A *time continuous Markov chain* is a sequence of random variables $X_t,\ t \in \mathbb{R}_0^+$ taking values of a finite set of states $\mathcal{A}$. X_{t_0} is distributed as $\pi^{(0)}$ and the Markov property holds:

$$Pr(X_{t_n} = x_n | X_{t_{n-1}} = x_{n-1}, ..., X_{t_0} = x_0) = Pr(X_{t_n} = x_n | X_{t_{n-1}} = x_{n-1})$$

for any $n \in \mathbb{N}$, time points $t_0 < t_1 < ... < t_n$ and any states $x_0, x_1, ..., x_n \in \mathcal{A}$.

The Markov chain is *time homogeneous* if there exists a *transition probability matrix* $P(t)$ such that

$$Pr(X_{s+t} = j | X_s = i) = P_{ij}(t), \quad \forall\ s, t \geq 0, \quad i, j \in \mathcal{A}.$$

The transition probability matrix $P(t)$ is a stochastic matrix and has the following properties:

- $P(0) = I$, $\qquad I$ - identity matrix,
- $P_{ij}(t) \geq 0$ and $\sum_j P_{ij}(t) = 1$,
- $P(s+t) = P(s)P(t)$ for $s, t \geq 0$ $\qquad$ (*Chapman-Kolmogorov equation*).

The time continuous Markov chain is *irreducible* if for any period $t > 0$ each state can reach each state: $P_{ij}(t) > 0 \quad \forall\ i, j \in \mathcal{A}$. In that case there exists a unique *stationary distribution* π which is the solution of $\pi P(t) = \pi$.

8.2.3 The Rate Matrix

We assume that the probability transition matrix $P(t)$ of a time continuous Markov chain is continuous and differentiable at any $t > 0$. I.e. the limit

$$\lim_{t \searrow 0} \frac{P(t) - I}{t} = Q$$

exists. Q is known as the *rate matrix* or the *generator* of the Markov chain. For very small time periods $h > 0$, transition probabilities are approximated by

$$\begin{aligned} P(h) &\approx I + hQ \\ P_{ij}(h) &\approx Q_{ij} \cdot h, \qquad i \neq j. \end{aligned}$$

From the last equation we see, that the entries of Q may be interpreted as substitution rate per site per year.

From the Chapman-Kolmogorov equation we get the forward and backward equation

$$\begin{aligned} \frac{d}{dt} P(t) &= \lim_{h \searrow 0} \frac{P(t+h) - P(t)}{h} \\ &= \lim_{h \searrow 0} \frac{P(t)P(h) - P(t)I}{h} \\ &= P(t) \lim_{h \searrow 0} \frac{P(h) - P(0)}{h} \\ \frac{d}{dt} P(t) &= P(t)Q = QP(t). \end{aligned}$$

This differential equation can be solved under the initial condition $P(0) = I$ and yields

$$P(t) = \exp(tQ) = \sum_{k=0}^{\infty} \frac{Q^k t^k}{k!}.$$

Thus, transition probabilities for any time $t > 0$ may be computed from the matrix Q. Q provides an infinitesimal description of the process. As $\sum_{j \in \mathcal{A}} P_{ij}(t) = 1$ we have

$$\sum_{j \in \mathcal{A}} Q_{ij} = 0$$

(the rows of the rate matrix sum to 0) and therefore $Q_{ij} \geq 0$ for $i \neq j$ and $Q_{ii} \leq 0$.

We may denote the time continuous Markov chain by $(\pi^{(0)}, Q)$ (if Q exists and has the above properties). Then π is a stationary distribution if

$$\sum_{i \in \mathcal{A}} \pi_i Q_{ij} = 0,$$

$$\pi Q = 0.$$

8.2.4 Definition of an Evolutionary Markov Process (EMP)

So far we have collected basic notions on time discrete and time continuous Markov chains. We additionally make the assumption that X_t is *reversible* and calibrate the rate matrix Q to a time unit. This enables us to define an *evolutionary Markov process (EMP)* according to Müller and Vingron [18]. The definiton of an EMP summarizes the requirements on the Markov chain such that it is suited to describe the substitution process at a site of a molecular sequence. We start by defining time units.

Definition. One *PEM* (percent of expected mutations) is the time one substitution event (mutation) per 100 sites is expected in. One *PAM* (percent of accepted mutations) is the time for which the average number of substitutions per 100 sites is one.

- Given a rate matrix Q one expects per time unit

  $$E := \sum_i \pi_i \sum_{j \neq i} Q_{ij} = -\sum_i \pi_i Q_{ii}$$

  substitutions per site. We calibrate Q by multiplying it with a constant. If $E = 1/100$, Q is calibrated to 1 PEM.

- Given a 1-step probability transition matrix $P^{(1)}$

  $$E' := \sum_i \pi_i \sum_{j \neq i} P^{(1)}_{ij} = 1 - \sum_i \pi_i P^{(1)}_{ii}$$

  is the average number of sites being in another state after one time unit. If $E' = 1/100$, $P^{(1)}$ is calibrated to 1 PAM.

In contrast to PAM units, PEM units take back mutations into account. Therefore 1 PEM is a slightly shorter time unit than 1 PAM.

We observe pairs of homologous sequences having evolved from a common ancestor. Yet we do not have any information about ancestral sequences. Therefore we assume that evolution from ancestors to descendants can be modeled by the same process as its reverse. Thus the divergence of two homologoues present day sequences is explained by one process. This property of Markov chains is called *reversibility.*

Definition. The Markov chain X_t is *reversible* if the probability for X_t being in state i at time $t = 0$ and reaching state j at time $t = s$ is the same as the probability for being in state j at time $t = 0$ and reaching state i at time $t = s$ for any $i, j \in \mathcal{A}$ and any $s > 0$. This requirement is equivalent to the *detailed balance equations*:

$$\pi_i P_{ij}(t) = \pi_j P_{ji}(t), \qquad \forall\ t > 0$$

$$\pi_i Q_{ij} = \pi_j Q_{ji}.$$

Now we are able to define an EMP.

Definition. We call a time continuous Markov chain X_t on the set of states $\mathcal{A}$ an *evolutionary Markov process (EMP)* with the stationary distribution π on the states if

- X_t is time homogeneous.
- X_t is stationary and the initial distribution $\pi^{(0)}$ is the stationary distribution π. Therefore X_t is distributed according to π for all $t \in \mathbb{R}_0^+$.
- X_t is irreducible: $P_{ij}(t) > 0$ for all $t > 0$ and $i, j \in \mathcal{A}$, i.e. π is unique.
- X_t is calibrated to 1 PEM: $\sum_i \pi_i Q_{ii} = -0.01$.
- X_t is reversible: $\pi_i P_{ij}(t) = \pi_j P_{ji}(t)$ for all $t > 0$ and $i, j \in \mathcal{A}$.

8.3 Nucleotide Substitution Models

In this section we focus on two important models for nucleotide substitutions, the Jukes-Cantor model and the Kimura-2-parameter model. Both models make assumptions on the rate matrix Q. Given Q the EMP is fully described as the stationary distribution π can be obtained by solving $\pi Q = 0$ with $\sum_i \pi_i = 1$.

8.3.1 The Jukes-Cantor model (JC)

The *Jukes-Cantor model* assumes that each substitution occurs at equal rate α. Thus the rate matrix is

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

The Jukes-Cantor model is reversible and the stationary distribution is the uniform distribution $\pi = (\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})$. Calibration to 1 PEM yields

$$-0.01 = \sum_{i=1}^{4} \pi_i Q_{ii} = -3\alpha,$$

$$\alpha = 1/300.$$

Due to the simple structure of Q, $\exp(tQ)$ can be calculated explicitly. The transition probability matrix is

$$P(t) = \begin{pmatrix} 1-3a_t & a_t & a_t & a_t \\ a_t & 1-3a_t & a_t & a_t \\ a_t & a_t & 1-3a_t & a_t \\ a_t & a_t & a_t & 1-3a_t \end{pmatrix},$$

where

$$a_t = \frac{1-\exp(-4\alpha t)}{4} = \frac{1-\exp(-4t/300)}{4}.$$

8.3.2 Jukes-Cantor correction

Given an alignment of two DNA sequences we want to estimate the evolutionary distance between the sequences. Let n denote the length of the alignment and u the number of mismatches. A naive distance estimator may take the relative amount of observed substitutions into account only, e.g. $D = u/n$. But D underestimates the amount of substitutions as multiple and back mutations may have been occured. Think of a nucleotide, e.g. 'A', being substituted by a 'G' and the 'G' being substituted by an 'A' again in the course of evolution. We cannot observe such an event. But we can calculate the probability that any nucleotide remains the same at any site in time t:

$$\begin{aligned} Pr(X_t = i|X_0 = i) &= \sum_{i \in \mathcal{A}} \pi_i P_{ii}(t) \\ &= 4 \cdot \frac{1}{4}(1 - 3a_t) \\ &= \frac{1 + 3\exp(-4\alpha t)}{4} \end{aligned}$$

We also know that 3α substitutions are expected per site and per time unit: $E = -\sum_i \pi_i Q_{ii} = 4 \cdot (3/4)\alpha = 3\alpha$. We denote the number of expected substitutions per 100 sites in time t by d: $d = 300\alpha t$ and therefore $\alpha t = d/300$. We can replace αt in the above equation and get

$$Pr(X_t = i|X_0 = i) = \frac{1 + 3\exp(-4d/300)}{4}.$$

We observe u mismatches at n sites and establish

$$Pr(X_t = i|X_0 = i) = \frac{n-u}{n} = 1 - D = \frac{1 + 3\exp(-4d/300)}{4}.$$

Solving for d yields

$$d = -\frac{300}{4} \ln\left(1 - \frac{4}{3}D\right) \text{ PEM.}$$

This is known as *Jukes-Cantor correction* (of the linear distance estimator D). If $d \ll n$, then $\ln(1 - (4/3)D) \approx -(4/3)D$ and $d \approx 100 \cdot D$ PEM. If there are more than five substitutions per 100 sites, that is $D > 0.05$, then $d > 100 \cdot D$ PEM. Note that if $D > 3/4$, d becomes undefined because the argument of the logarithm becomes negative.

8.3.3 Kimura 2-parameter model (K2P)

Transitions ($A \leftrightarrow G$ and $C \leftrightarrow T$) are more frequently observed than transversions as A and G are purines and C and T are pyrimidines. The *Kimura 2-parameter model* takes that into account by introducing different rates for transitions (α) and transversions ($\beta < \alpha$). With the order AGCT of nucleotides, the rate matrix Q is

$$Q = \begin{pmatrix} -\alpha - 2\beta & \alpha & \beta & \beta \\ \alpha & -\alpha - 2\beta & \beta & \beta \\ \beta & \beta & -\alpha - 2\beta & \alpha \\ \beta & \beta & \alpha & -\alpha - 2\beta \end{pmatrix}.$$

The stationary distribution is also uniform and calibration to 1 PEM yields $\alpha + 2\beta = 1/100$. The probability transition matrix is

$$P(t) = \begin{pmatrix} 1-(a_t+2b_t) & a_t & b_t & b_t \\ a_t & 1-(a_t+2b_t) & b_t & b_t \\ b_t & b_t & 1-(a_t+2b_t) & a_t \\ b_t & b_t & a_t & 1-(a_t+2b_t) \end{pmatrix},$$

where

$$a_t := (2E_t - e_t)/4, \qquad E_t := 1 - \exp(-2t(\alpha+\beta)),$$
$$b_t := e_t/4, \qquad e_t := 1 - \exp(-4t\beta).$$

8.4 Modeling Amino Acid Replacements

There are only four states in Markov chains describing the evolution of DNA sequences, the four nucleotides. The models presented in the previous sections make simple assumptions on the rate matrices Q. This kind of procedure is not suited when modeling protein evolution. There are 20 amino acid residues with a variety of replacement frequencies being distributed irregularly. In general one wants to estimate the rate matrix Q or the probability transition matrix P by means of amino acid sequence alignments.

8.4.1 Parameter estimation

The strategy of Dayhoff et al. (1978) is to estimate the 1-step probability transition matrix $P^{(1)}$ of a time discrete Markov chain and to extrapolate to higher PAM distances. For this purpose pairwise alignments of closely related sequences are collected which are assumed to be correct and which contain approximately 1% mismatches. m_{ij} holds the frequencies a pair of residues (i, j) occurs with, where each pair of residues is counted twice ((i, j) and (j, i)) such that the Markov chain is reversible. f_i denotes the frequency the residue i occurs and N holds the number of residue pairs. The estimator is based on the equation

$$\frac{m_{ij}}{N} = f_i \cdot P^{(1)}_{ij}.$$

$P^{(1)}$ is calibrated to 1 PAM and the k-step transition matrices are obtained by $P^{(k)} = P^k = PP^{k-1}$, for $k \geq 2$.

The main disadvantage of this method is that closely related sequences are taken into account only. It is desirable to exploit sequence alignments of different evolutionary distances for parameter estimation. Say we want to estimate the rate matrix Q of an EMP. The problem is that the estimator for Q clearly should account for the evolutionary divergence of each alignment in the dataset. But evolutionary divergence of an alignment has to be estimated by means of the model parameters Q. Müller and Vingron [18] present an iterative approach cycling between estimating evolutionary distances of sequences in an alignment and updating the current rate matrix Q.

8.4.2 Score matrices

Searching a protein database with a query protein requires a similarity measure on amino acid sequences. Usually similarity measures on amino acid sequences are based on a similarity measure on the residues which is stored in a score matrix (S_{ij}). S assigns a score to each pair of residues (i, j). The PAM matrices are based on a Markov model. The rationale is the following: Similar residues are replaced by each other more frequently than less similar residues. Vice versa one defines similarity by means of replacement frequencies and takes the distribution of residues into account. Consider the following ratio:

$$\frac{\pi_i P_{ij}(t)}{\pi_i \pi_j} = \frac{\pi_j P_{ji}(t)}{\pi_i \pi_j}$$

The numerator is the probability to observe the pair (i, j) in a pair of sequences which have evolved according to the model with the transition matrix $P(t)$. The denominator is the probability to observe the pair (i, j) in independent sequences. The *score* is defined as the logarithm of this ratio,

$$S_{ij}(t) := \ln \frac{\pi_i P_{ij}(t)}{\pi_i \pi_j}.$$

The score is positive if the pair (i, j) frequently occurs in evolutionarily related sequences.

8.4.3 ML-Estimation of evolutionary distances

Given two sequences $A = (A_1, ..., A_n)$ and $B = (B_1, ..., B_n)$ of the same length which have evolved according to an EMP with rate matrix Q. We want to estimate the evolutionary distance between A and B in PEM units. The probability that A and B have evolved from each other in t time units is

$$Pr(A, B; t) := \prod_{k=1}^{n} \pi_{A_k} \, P_{A_k, B_k}(t).$$

We consider $Pr(A, B; t)$ as likelihood function of time t to be estimated and try to find a t such that the probability for the observed data A and B is maximal (Maximum Likelihood Estimator). Maximizing the logarithm yields the same t. Note that the terms π_{A_k} do not depend on t. Therefore we try to find the maximum of the function

$$\mathcal{L}(t) := \sum_{k=1}^{n} \log\left(P_{A_k, B_k}(t)\right).$$

In general this is done numerically.

8.5 Example: Maximum Likelihood and the Hardy-Weinberg Equilibrium

In this section we present another example of maximum likelihood estimation, the statistical analysis of the Hardy-Weinberg Disequilibrium in population genetics. The section is based on [19], Section 8.5.1.

First recall the definition of the multinomial distribution: Perform n random experiments, each of which has one of m different outcomes with the respective probabilities $p_1, \ldots, p_m$. (Example: roll a die n times: $m = 6$.) Then let X_i be the number of outcomes of type i. Of course, the X_i are not independent, since they have to sum up to n. The distribution function of the multinomial distribution can (more or less straightforwardly) be found as

$$P(x_1, \ldots, x_m) \;=\; \frac{n!}{\prod_{i=1}^{m} x_i!} \prod_{i=1}^{m} p_i^{x_i}.$$

Now we want to condition this on the probabilities p_i, hence we have

$$P(x_1,\dots,x_m \mid p_1,\dots,p_m) \;=\; \frac{n!}{\prod_{i=1}^m x_i!}\prod_{i=1}^m p_i^{x_i}.$$

The likelihood function (joint density function as a function of the model parameters $p_1,\dots,p_m$), $L := P(\text{data} \mid \text{model})$ follows:

$$L(p_1,\dots,p_m) \;=\; \frac{n!}{\prod_{i=1}^m x_i!}\prod_{i=1}^m p_i^{x_i}.$$

The log-likelihood hence is

$$\mathcal{L}(p_1,\dots,p_m) \;=\; \log L(p_1,\dots,p_m) \;=\; \log n! - \sum_{i=1}^m \log x_i! + \sum_{i=1}^m x_i \log p_i.$$

Now to the Hardy-Weinberg Equilibrium: Assume a locus with two different alleles, A and a, such that the three different genotypes AA, Aa, and aa are possible. According to Hardy and Weinberg, in equilibrium these three genotypes occur in a population with frequencies $p_1 = (1-p)^2$, $p_2 = 2p(1-p)$, and p^2, for some value of p.

Example. In a study on blood types, the following counts were measured: $x_1 = 342[AA]$, $x_2 = 500[Aa]$, $x_3 = 187[aa]$, such that $n = 1029$.

Questions:

1. What is the maximum likelihood estimator (function)?
2. What is the maximum likelihood estimate under these data, i.e. which value of p has the highest likelihood?

Answer:

$$\begin{aligned}\mathcal{L}(p) \;=\; l(p_1,p_2,p_3) \;&=\; \log n! - \sum_{i=1}^3 \log X_i! + X_1\log(1-p)^2 + X_2\log 2p(1-p) + X_3\log p^2\\ &=\; \log n! - \sum_{i=1}^3 \log X_i! + (2X_1+X_2)\log(1-p) + (2X_3+X_2)\log p + X_2\log 2\end{aligned}$$

(Need not explicitly incorporate that the probabilities sum to 1.)

Setting the derivative equal zero yields

$$\frac{d}{dp}\mathcal{L}(p) \;=\; -\frac{2X_1+X_2}{1-p} + \frac{2X_3+X_2}{p} \;=\; 0$$

which can be solved as follows:

$$\begin{aligned}\hat{p} \;&=\; \frac{2X_3+X_2}{2X_1+2X_2+2X_3}\\ &=\; \frac{2X_3+X_2}{2n} \qquad \text{(maximum likelihood estimator)}\\ &=\; \frac{2\times 187+500}{2\times 1029} \;=\; .4247 \qquad \text{(maximum likelihood estimate)}\end{aligned}$$

It remains the question, how precise is such an estimate? This can also be estimated, for example by applying the bootstrapping method, see Section 10.

Chapter 9

Maximum Likelihood Trees

9.1 Computing the Likelihood of a Given Tree

In the context of reconstructing phylogenetic trees, the likelihood function which under a fixed model returns for given data the likelihood that these data were produced under the model, $L(\text{data}) := P(\text{data} \mid \text{model})$, can be written as

$$L(\text{alignment}) = P(\text{alignment} \mid \text{tree}).$$

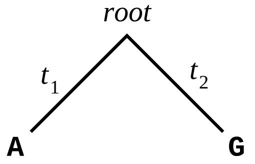
In the following we study a method to reconstruct the most likely tree for a given multiple sequence alignment, under a given evolutionary model.

Observations:

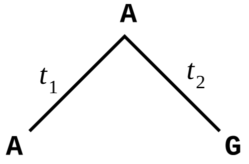
1. The tree with the highest likelihood can be found by computing the likelihood for each possible tree topology and then choosing the one with the highest likelihood value.
2. The likelihood can be computed for each site of the alignment independently. The total likelihood is then the product over all characters (alignment positions).

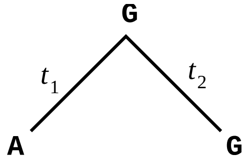
Hence for the moment we will assume that we have given a fixed tree topology with a single character state at each leaf and an evolutionary model that provides us for two character states X and Y and an evolutionary distance t with a probability $P_{XY}(t)$ that X has changed to Y within time t.

A simple example:



Assuming the reduced alphabet $\mathcal{A} = \{A, G\}$, the total likelihood of this tree is the sum of the likelihoods for the two possible assignments of character states at the root:





$$\begin{aligned} L_{\text{total}} &= L_{A \text{ at } root} + L_{G \text{ at } root} \\ &= P(A \text{ at } root) P_{AA}(t_1) P_{AG}(t_2) + P(G \text{ at } root) P_{GA}(t_1) P_{GG}(t_2). \end{aligned}$$

$P(X \text{ at } root)$ is often chosen as the background frequency π_X of state (DNA base) X. Often log-likelihoods are used for easier computation.

After the relation between likelihood and branch lengths is established, the branch lengths t_i can be adjusted. In this simple example, a short calculation allows to maximize the likelihood using $\frac{\partial}{\partial t_i} \ln L = 0$. In larger examples, the maximum is usually computed numerically, e.g. by Newton's method.

The general method: To compute the likelihood for a larger tree, we use a dynamic programming algorithm. For each vertex v, we define the *conditional likelihood* $L_X(v)$ as the likelihood of the subtree below v given that the character state in v is X. Then, as above, we have given a tree labelled with one character state at each leaf and an evolutionary model. The algorithm goes as follows.

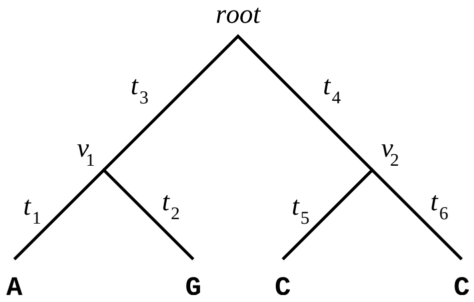
1. Choose an arbitrary root.
2. Traverse the tree bottom-up from the leaves to the root and do the following:
   a) Assign the likehood at the leaves: For each leaf v, $L_X(v) = \delta_{XY}$ where Y is the character at leaf v.
   b) At an internal node v compute all conditional likelihoods $L_X(v)$. Therefore the likelihoods of the different branches are multiplied:

$$L_X(v) = \prod_{v' \text{ child of } v} \sum_{Y \in \mathcal{A}} P_{XY}(t_{v \to v'}) L_Y(v').$$

3. The total likelihood of the tree is the sum of all conditional likelihoods at the root, weighted by the background probability π_X of the respective character state (DNA base) X:

$$L = \sum_{X \in \mathcal{A}} \pi_X L_X(root).$$

A larger example: The following figure shows a rooted tree with four leaves.



Then, for example,

$$L_A(v_1) = P_{AA}(t_1)P_{AG}(t_2)$$

or, more general for the root,

$$L_X(root) = \left(\sum_Y P_{XY}(t_3)L_Y(v_1)\right)\left(\sum_Y P_{XY}(t_4)L_Y(v_2)\right).$$

The total likelihood is:

$$L = \pi_A L_A(root) + \pi_C L_C(root) + \pi_G L_G(root) + \pi_T L_T(root)$$

Adjusting branch lengths: An algorithm to adjust branch lengths in the general case is the following.

Repeat for all branches several times:

1. Choose an edge.
2. Choose a node incident to this edge as the root.
3. Compute the maximum likelihood for the rest of the tree
4. Choose the branch length such that the likelihood of the whole tree is maximized.

Notes on the Maximum Likelihood method:

- Under different models, different trees can give the maximum likelihood.
- Maximum likelihood is in spirit similar to maximum parsimony, but (1) the cost of a change in parsimony is not a function of the branch length; (2) maximum parsimony only looks at the single, lowest cost solution, whereas maximum likelihood looks at the combined likelihood for all solutions (ancestral states) consistent with the tree and branch lengths.
- The method is extremely time consuming. Heuristic methods exist, e.g. Quartet Puzzling: For each quartet (group of 4 sequences), (1) consider all of the three possible tree topologies and compute their likelihood, (2) compose intermediate trees from the quartet trees (repeat multiple times), and (3) construct a majority rule consensus tree from all the intermediate trees; then optimize the branch lengths. For details on the Quartet Puzzling method, see Chapter 12.

9.2 Consistency

There is an old discussion about which method for reconstructing phylogenetic trees is the best one. Without going too much into the details, here is a list of possible criteria to compare the different methods:

- *Consistency*: The tendency of the method to converge on the correct answer as more data (characters) are added.
- *Efficiency* (or *power*): the ability of the method to recover the correct answer with limited amounts of data.
- *Robustness*: A method is robust if it is relatively insensitive to violations of its assumptions.
- *Computational speed*: The length of time a method takes to solve a problem.

- *Discriminating ability*
- *Versatility*: What kind of information can be incorporated into the analysis?

In the following we want to focus on consistency, which has received much attention, although it also has its weaknesses. Our discussion of consistency is based on Huelsenbeck: Performance of Phylogenetic Methods in Simulation, Systematic Biology 44(1): 17-48, 1995 and Hillis, Mable, Moritz, Chapter 12 in Hillis, Moritz, Mable: Molecular Systematics, 1996.

Definition: A method is *consistent* if it converges on the correct tree when given infinite data.

All methods are consistent when their assumptions are met, and all methods are inconsistent if their assumptions are sufficiently violated. Therefore one has to specify the conditions under which a method is consistent.

For example, most distance-based methods (except UPGMA) are consistent under the Jukes-Cantor model. Maximum parsimony can be made consistent by using a Hadamard transformation [16] to correct the data.

The notion of consistency is not necessarily useful to assess the practical value of a method: A method can be consistent, but very inefficient (like Lake's method of invariants) which means that in practice one will never be able to collect enough data to get a good result with high likelihood.

To test the accuracy of a method in practice, one can apply the method to real (experimentally verified) or numerical (simulated) data. Experimental studies are expensive and time consuming. Simulations are cheap and fast, but might be problematic (model realism, simulation bias).

Part V

Advanced Topics

Chapter 10

Assessing the Quality of Reconstructed Trees

Once a phylogenetic tree is constructed, it is important to ensure that the result is not the result of some algorithmic artefacts. For example, remember the discussion about methods that always reconstruct a tree, even if the underlying data is not at all tree-like. In such a case, the result may be some tree, but if the data were looking only slightly different, the tree could be rather different.

One way towards solving this problem is the Splits decomposition method presented in Chapter 7 that produces a tree only for tree-like data, and otherwise a network that shows how strong and in which region the data deviate from a tree.

A different way to assess the reliability of a reconstructed phylogenetic tree is to apply the bootstrap, which is explained in the first part of this chapter. The second part describes ideas for combining ("puzzling") several quartet trees into one large tree whose edges are annotated by the support they find in the qurtets. A third way to judge about the tree-likeness of data, also using quartets, is *likelihood mapping*.

10.1 Bootstrapping

The standard reference for the Bootstrap method is the book by Efron and Tibshirani [8]. We first explain the general method, and then how it can be applied to phylogenetic tree methods.

10.1.1 The General Idea of Bootstrapping

The general idea of bootstrapping is the following: Let

$$\mathcal{X}_n = (X_1, X_2, \ldots, X_n)$$

be a vector of n samples drawn from an unknown distribution. Let $T = T(\mathcal{X})$ be some statistic on the sample, e.g. the median. Since the statistic depends on the actual sample (it would be different with a new sample), it can be seen as a random variable, following the sample statistic. In order to estimate the error of the statistic, we would like to know its standard error. In principle, if the underlying distribution of $\mathcal{X}$ was known, it would be possible to (a) increase n or (b) re-draw $\mathcal{X}$.

Usually the true distribution is unknown, though, and hence the following trick is used, called *bootstrapping*: Use the sample data $\mathcal{X}$ and draw from them n samples with repetition:

$$\mathcal{X}_n^* = (X_1^*, X_2^*, \ldots, X_n^*).$$

This can be repeated several times, giving B *bootstrap replicates* $\mathcal{X}_n^{*b}$ for $b = 1, \ldots, B$. For each replicate, the statistic is computed. This allows, e.g., to compute a standard error.

A typical example is a (usually small) set of treatment and control data from a medical experiment (where true replicates are usually very expensive).

10.1.2 Bootstrapping in the Phylogenetic Context

In the context of phylogenetic tree reconstruction, bootstrapping was introduced by Felsenstein [9].

Here the sample data are the columns of a multiple alignment. Resampling is done on columns of the alignment. Each replicate is again a multiple alignment of n columns, but some columns may be duplicated, others may be missing. Each of these alignments gives rise to a tree. The various trees may be different.

Note that bootstrapping is possible for any tree reconstruction method.

Often the bootstrap replicates are used to measure the support for the different parts of the original tree, i.e. the tree T that was created from the original alignment. A simple way to do this is the following. Each edge in T defines a split of the taxa. Then one counts for each split (edge) of T, in how many of the bootstrap replicates this split is also realized. This value, often written as a percentage value, is called the *bootstrap support* of the edge.

10.2 Likelihood Mapping

1. Four sequences. Given four sequences, let L_i be the maximum likelihood for topology i. Define $p_i = L_i/(L_1+L_2+L_3)$. The p_i define a point in the two-dimensional simplex (for example, illustrated by an equilateral triangle). Each corner corresponds to a topology. See Figures 2 and 3 in [23].

2. The general case. There are $\binom{n}{4}$ quartets. Draw all or a large number of the points. The distribution of points in the resulting picture shows the tree-likeness of the data set. If many points are not in the corners, the data is star-like or not tree-like at all. (Note: The opposite is not generally true.) Figure 4 in [23] shows interesting simulation results for star-like and tree-like data.

Chapter 11

Consensus trees and consensus methods

11.1 The Strict Consensus Tree

Given several trees with the same set of elements at their leaves, it is often desirable to compute an average or *consensus* tree of these trees. The method described here is called the *strict consensus tree.* Here, one collects all splits that have more than 50% support from the given trees.

Theorem. These splits form a tree.

Proof: 1) The splits chosen in this way are all pairwise compatible. To see this, assume that there is a pair of splits that is not compatible. Then one can easily derive a contradiction.

2) Remember that for binary characters, pairwise compatibility implies that there exists a perfect phylogeny (2nd version of Gusfield's theorem in Section 3.3), hence they form a (possibly multifurcating) tree. ☐

Definition. The tree that realizes all these splits with more than 50% support is called the *strict consensus tree.*

Sometimes the strict consensus tree contains highly multifurcating vertices. Therefore it may be further refined by (greedily) adding all splits that fit into the tree and annotating the edges with support values.

11.2 Bayesian methods

Chapter 12

Parent trees and supertree methods

12.1 Quartet Puzzling

A fast method to compute trees that approximate the maximum likelihood tree is *Quartet Puzzling* [22]. It contains two interesting steps where different trees are compared and joined together.

The procedure is the following:

1. Compute the maximum likelihood solution for all quartets of the data set. (Any other tree reconstrction method for four taxa would work as well.)
2. Piece together the optimal quartets, given an order: one taxon after the other is inserted. Let E be the next taxon to be inserted. For each edge count the number of quartets containing E that imply E to be inserted *not* in this edge. The edge with the lowest overall count wins. (See Figure 2 in [22].)
3. Since the method of Step 2 is order-dependent, it is repeated several times for different input orders. The resulting trees are combined into a so-called strict consensus tree by the method described below.

Quartet puzzling often yields trees rather similar to the maximum likelihood tree. The method, while still slow for many taxa, is much faster than computing the exact maximum likelihood tree. For faster computations, there exists a parallelized version of quartet puzzling.

12.2 NeighborNet

The following is from [4].

Combination of Neighbor Joining and SplitsTree

- Incremental tree construction similar to NJ.
- Result is a network as in SplitsTree, but with a better resolution
- $O(n^3)$ running time

Bibliography

[1] K. Atteson. The performance of the neighbor joining method of phylogeny reconstruction. In B. M. et al., editor, *Mathematical Hierarchies and Biology*, pages 133–148. American Mathematical Society, 1997.

[2] H.-J. Bandelt and A. W. M. Dress. A canoncal decomposition theory for metrics on a finite set. *Adv. Math.*, 92:47–105, 1992.

[3] H.-J. Bandelt and A. W. M. Dress. Split decomposition: A new and useful approach to phylogenetic analysis of distance data. *Mol. Phylogenet. Evol.*, 1:242–252, 1992.

[4] D. Bryant and V. Moulton. NeighborNet: An agglomerative method for the construction of planar phylogenetic networks. In R. Guigó and D. Gusfield, editors, *Proceedings of the Second International Workshop on Algorithms in Bioinformatics, WABI 02*, volume 2452 of *LNCS*, pages 375–391, Berlin, 2002. Springer Verlag.

[5] P. Bunemann. The recovery of trees from measures of dissimilarity. In J. H. et al., editor, *Mathematics and the Archeological and Historical Sciences*, pages 387–359. Edinbhurg University Press, 1971.

[6] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, MA, 1989.

[7] R. Desper and O. Gascuel. Fast and accurate phylogeny reconstruction algorithms based on the minimum-evolution principle. In R. Guigó and D. Gusfield, editors, *Proceedings of the Second International Workshop on Algorithms in Bioinformatics, WABI 02*, volume 2452 of *LNCS*, pages 357–374, Berlin, 2002. Springer Verlag.

[8] B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap.* CRC Press, Boca Raton, FL, 1998.

[9] J. Felsenstein. Confidence limits on phylogenies: An approach using the bootstrap. *Evolution*, 39(4):783–791, 1985.

[10] J. Felsenstein. *Inferring Phylogenies.* Sinauer Associates, Inc., Sunderland, Massachusetts, 2004.

[11] W. Fitch and E. Margoliash. Construction of phylogenetic trees. *Science*, 155:279–284, 1967.

[12] L. R. Foulds and R. L. Graham. The Steiner problem in phylogeny is NPcomplete. *Adv. Appl. Math.*, 3:43–49, 1982.

[13] A. J. Griffiths, J. H. Miller, D. T. Suzuki, R. C. Lewontin, and W. M. Gelbart. *An Introduction to Genetic Analysis (6th ed.).* W.H. Freeman and Company, New York, 1996.

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

[15] D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, New York, NY, 1997.

[16] M. D. Hendy and D. Penny. Spectral analysis of phylogenetic data. *J. Classif.*, 10:5–24, 1993.

[17] C. D. Michener and R. R. Sokal. A quantitative approach to a problem in classification. *Evolution*, 11:130–162, 1957.

[18] T. Müller and M. Vingron. Modeling aamino acid replacement. *J. Comp. Biol*, 6:761–776, 2000.

[19] J. A. Rice. *Mathematical Statistics and Data Analysis.* Duxbury Press, (somewhere), 1995.

[20] D. Sankoff. Minimal mutation trees of sequences. *SIAM J. Appl. Math.*, 28:35–42, 1975.

[21] B. Schwikowski and M. Vingron. The deferred path heuristic for the generalized tree alignment problem. *J. Comp. Biol.*, 4:415–431, 1997.

[22] K. Strimmer and A. von Haeseler. Quartet puzzling: A quartet maximum-likelihood method for reconstructing tree topologies. *Mol. Biol. Evol.*, 13(7):964–969, 1996.

[23] K. Strimmer and A. von Haeseler. Likelihood-mapping: A simple method to visualize phylogenetic content of a sequence alignment. *Proc. Natl. Acad. Sci. USA*, 94:6815–6819, 1997.

[24] J. Studier and K. Keppler. A note on the neighbor-joining algorithm of saitou and nei. *Molecular Biology and Evolution*, 5:729–731, 1988.

[25] M. Vingron and A. von Haeseler. Towards integration of multiple alignment and phylogenetic tree construction. *J. Comp. Biol.*, 4:23–34, 1997.

[26] T. Warnow. Tree compatibility and inferring evolutionary history. *Journal of Algorithmics*, 16:388–407, 1994.

[27] M. S. Waterman, T. F. Smith, M. Singh, and W. A. Beyer. Additive evolutionary trees. *J. Theor. Biol.*, 64:199–213, 1977.