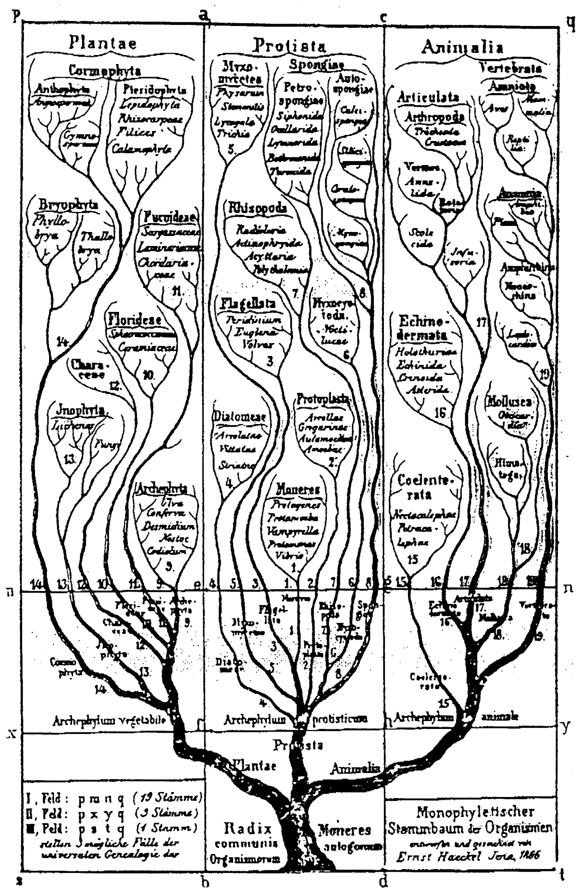
5 Phylogeny



Evolutionary tree of organisms Ernst Haeckel, 1866

Sources for parts of this Chapter:

- R. Durbin, S. Eddy, A. Krogh & G. Mitchison, Biological sequence analysis, Cambridge, 1998
- J. Setubal & J. Meidanis, Introduction to computational molecular biology, 1997.
- D.W. Mount. Bioinformatics: Sequences and Genome analysis, 2001.
- D.L. Swofford, G.J. Olsen, P.J.Waddell & D.M. Hillis, Phylogenetic Inference, in: D.M. Hillis (ed.), Molecular Systematics, 2 ed., Sunderland Mass., 1996.

5.1 Phylogenetic analysis

Given a collection of extant species. The goal of phylogenetic analysis is to determine and describe the *evolutionary relationship* between the species. In particular, this involves determining the order of speciation events and their approximate timing.

It is generally assumed that *speciation* is a branching process: a population of organisms becomes separated into two sub-populations. Over time, these evolve into separate species that do not cross-breed.

Because of this assumption, a *tree* is often used to represent a proposed phylogeny for a set of species, showing how the species evolved from a common ancestor. We will study this in detail.

However, there are a number of situations in which a tree is less appropriate than a phylogenetic *network*. We will study this in a later Chapter.

5.2 Phylogenetic trees

In the following, we will use $X = \{x_1, x_2, \ldots, x_n\}$ to denote a set of *taxa*, where a *taxon* x_i is simply a representative of a group of individuals defined in some way.

A *phylogenetic tree* (on X) is a system $T = (V, E, \lambda)$ consisting of a connected graph (V, E) without cycles, together with a *labeling* λ of the leaves by elements of X, such that:

1. every leaf is labeled by exactly one taxon, and
2. every taxon appears exactly once as a label.

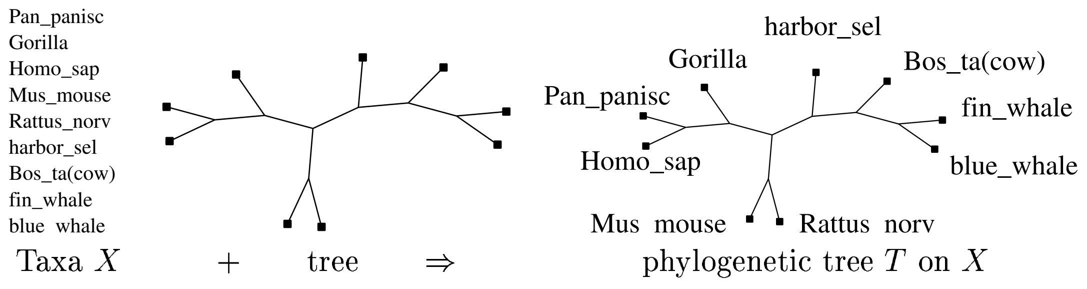
Further, we require that either all internal nodes have degree ≥ 3, in which case T is called *unrooted*, or there exists precisely one internal *root* node ρ of degree 2, and T is called *rooted.*

A phylogenetic tree T is called *binary*, if every internal node v has degree 3, except ρ, if T is rooted.

Sometimes we will relax the definition to allow labels to be placed on internal nodes, or nodes to carry multiple labels. Occasionally we will allow an arbitrary node to be root.

5.3 Unrooted trees

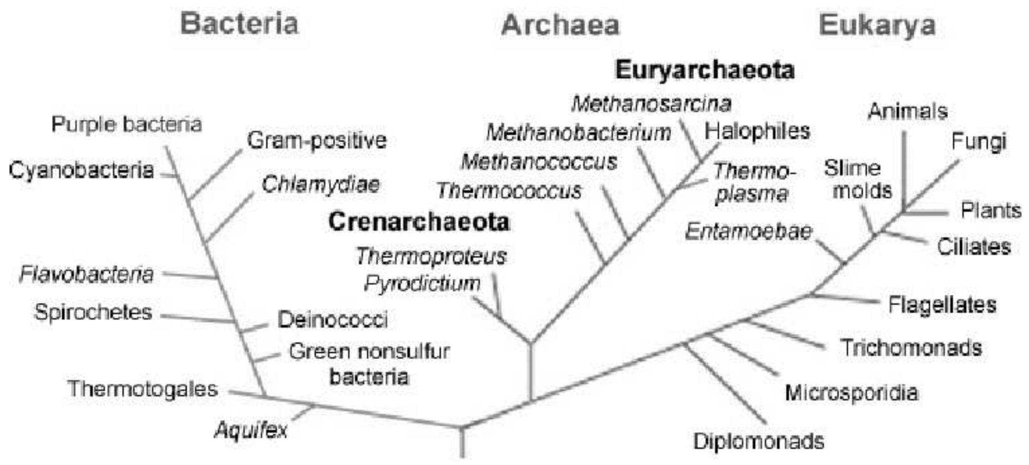
An unrooted phylogenetic tree is obtained by placing a set of taxa on the leaves of a tree:



Unrooted trees are often displayed using this type of *circular* layout.

5.4 Rooted trees

A rooted tree is usually drawn with the root placed at the bottom, top or left of the figure:



Modern version of the tree of life.

5.5 Edge lengths

Consider a phylogenetic tree T on X. The *topology* of the tree describes the putative order of speciation events that gave rise to the extant taxa.

Additionally, we can assign *lengths* to the edges of the tree. Ideally, these lengths should represent the amount of time that lies between two speciation events. However, in practice the edge lengths usually represent quantities obtained by some given computation and only correspond very indirectly to time.

In the following, we will use $\omega : E \to \mathbb{R}_{\geq 0}$ to specify edge lengths and will use $T = (V, E, \lambda, \omega)$ to denote a phylogenetic tree with edge lengths.

5.6 Computer representation of a phylogenetic tree

Let X be a set of taxa and T a phylogenetic tree on X.

To represent a phylogenetic tree in a computer we maintain a set of nodes V and a set of edges E. Each edge $e \in E$ maintains a reference to its source node $s(e)$ and target node $t(e)$. Each node $v \in V$ maintains a list of references to all adjacent edges.

Each node $v \in V$ maintains a reference $\lambda(v)$ to the taxon that it is labeled by, and, vice versa, $\nu(x)$ maps a taxon x to the node that it labels.

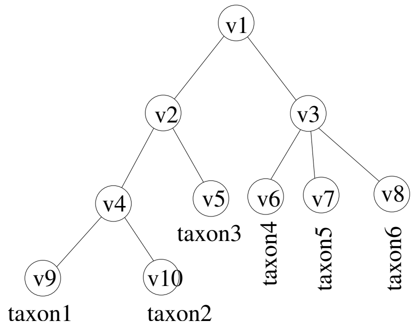
Each edge $e \in E$ maintains its length $\omega(e)$.

5.7 Nested structure

A rooted phylogenetic tree $T = (V, E, \lambda)$ is a *nested* structure: Consider any node $v \in V$. Let T_v denote the subtree rooted at v. Let $v_1, v_2, \ldots, v_k$ denote the children of v. Then T_v is obtainable from its subtrees $T_{v_1}, T_{v_2}, \ldots, T_{v_k}$ by connecting v to each of their roots.

Such a nested structure can be written using nested brackets:

Phylogenetic tree



nested description

$$v_1$$
$$\downarrow$$
$$(v_2, v_3)$$
$$\downarrow$$
$$((v_4, v_5), (v_6, v_7, v_8))$$
$$\downarrow$$
$$(((v_9, v_{10}), v_5), (v_6, v_7, v_8))$$

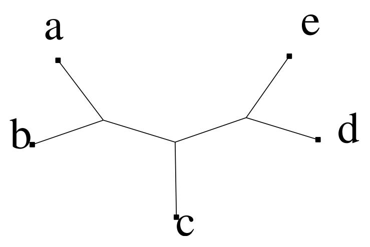
Description: $(((taxon_1, taxon_2), taxon_3), (taxon_4, taxon_5, taxon_6))$

5.8 Printing a phylogenetic tree

Phylogenetic trees are usually printed using the so-called *Newick* format which uses pairs of brackets to indicate the hierarchical structure of a tree. For example,

`((a,b),c,(d,e));`

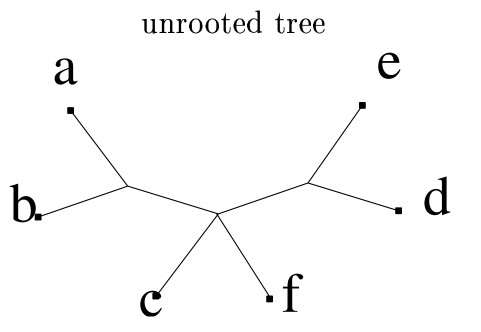
describes the following unrooted phylogenetic tree, without edge lengths:

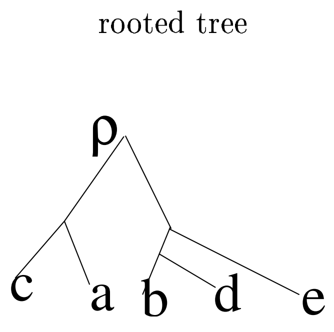


The following algorithm recursively prints a tree in Newick format. It is initially called with $e = null$ and v set to ρ, if the tree is rooted, or v set to an arbitrary internal node, else.

```
**Algorithm** toNewick($e, v$)
Input: Phylogenetic tree $T = (V, E)$ on $X$, with labeling $\lambda : V \to X$
Output: Newick description of tree
begin
if $v$ is a leaf then
        print $\lambda(v)$
else // $v$ is an internal node
        print '('
        for each edge $f \neq e$ adjacent to $v$ do
                If this is not the first pass of the loop, print ','
                Let $w \neq v$ be the other node adjacent to $f$
                call toNewick($f, w$)
        print ')'
end
```

Here are two examples:





Newick string:
`((a,b),c,f,(e,d))`
or e.g.:
`((a,b),(c,f,(e,d)))`

Newick string:
`((c,a),((c,d),e))`
...

5.9 Parsing a phylogenetic tree

We need to be able to read a phylogenetic tree into a program. The following algorithm parses a tree in Newick format from a (space-free) string $s = s_1 s_2 \ldots s_m$. The initial call is parseNewick($1, m, null$).

```
Algorithm parseNewick($i$,$j$,$v$)
Input: a substring $s_i \ldots s_j$ and a root node $v$
Output: a phylogenetic tree $T = (V, E, \lambda)$
begin
while $i \leq j$ do
        Add a new node $w$ to $V$
        if $v \neq null$ then add a new edge $\{v, w\}$ to $E$

        if $s_i$ = '(' then // recurse on subtree
              Set $k$ to the position of the balancing close-bracket for $i$
        call parseNewick($i+1, k-1, w$) // strip brackets and recurse

    else // hit a leaf
        Set $k = \min\{k \geq i \mid s_{k+1} = \text{','} \text{ or } k+1 = j\}$
        Set $\lambda(w) = s_i \ldots s_k$ // grab label for leaf

    Set $i = k+1$ // advance to next ',' or $j$
    if $i < j$ then
        Check that $i+1 \leq j$ and $s_{i+1} = \text{','}$
        Increment $i$ // advance to next token
end
```

In the Newick format, each pair of matching brackets corresponds to an internal node and each taxon label corresponds to an internal node. To add edge lengths to the format, specify the length *len* of the edge along which we visited a given node by adding *:len* behind the closing bracket, in the case of an internal node, and behind the label, in the case of a leaf.

For example, consider the tree `(a,b,(c,d))`. If all edges have the same length 1, then this tree is written as `(a:1,b:1,(c:1,d:1):1)`.

5.10 Drawing a phylogenetic tree

An *embedding* of a phylogenetic tree is given by an assignment of coordinates $(x(v), y(v))$ to each node v. Any edge e is represented by the line segment connecting the points $(x(v), y(v))$ and $(x(w), y(w))$ associated with the two nodes v and w adjacent to e. We will require that following additional conditions are satisfied:

1. no two line segments representing edges cross, and

2. for every edge e, the length of the line segment representing e is $\omega(e)$.

We will now discuss how to compute an embedding for an arbitrary unrooted phylogenetic tree. The algorithm proceeds in two stages. In the first stage, a direction is recursively computed for each edge. Then, in the second stage, all nodes are initially placed at the same point and from there they they are moved in the directions computed for the edges.

5.11 Circular tree embedding

Let T be a phylogenetic tree, together with an embedding $\epsilon : V \to \mathbb{R}^2$. Choose any leaf r and call it the *reference leaf*. Starting at r, circumnavigate the tree in anti-clockwise order and give each leaf an *index* $h(v) = 0, 1, \ldots, n-1$, starting with $h(r) = 0$.

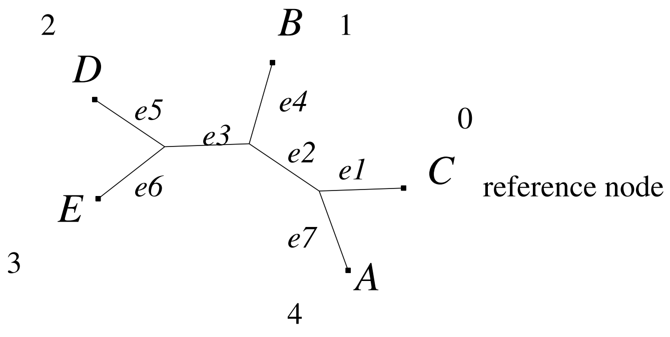
Direct all edges e of T away from r and let $V(e)$ denote the set of all leaves reachable from e in the directed graph.

We say that the embedding ϵ is a *circular embedding* with reference leaf r, if for any directed edge e we have that its angle equals

$$\alpha(e) = \frac{2\pi}{n|V(e)|} \sum_{v \in V(e)} h(v).$$

In other words, the leaves are placed around the unit circle at positions $\frac{2\pi}{n}, 1\frac{2\pi}{n}, \ldots, n-1\frac{2\pi}{n}$ and a edge e points in the average direction of the set of leaves that it separates the reference node r from.

Example:



This is a circular layout with reference node C, which is verified as follows:

- $\alpha(e_1) = \frac{2\pi}{5\cdot4}(1+2+3+4) = \pi$,
- $\alpha(e_2) = \frac{2\pi}{5\cdot3}(1+2+3) = \frac{4}{5}\pi$,
- $\alpha(e_3) = \frac{2\pi}{5\cdot2}(2+3) = \pi$,
- ...

5.12 Computing the edge angles

Now, given a phylogenetic tree T, how do we obtain a circular embedding for it? We first assign an angle to each edge. Then, using these angles and the given edge lengths, we will assign coordinates to each node.

The following algorithm visits all edges and assign an angle to each. The initial call is setAngles$(0, null, r)$, where r is the reference leaf.

```
**Algorithm** setAngles($h, e, v$)
Input: Number $h$ of nodes placed, arrival edge $e$, current node $v$
Output: Angle $\alpha(e)$ for every edge $e \in E$.
begin
if $v$ is a leaf then
    return $h + 1$
```

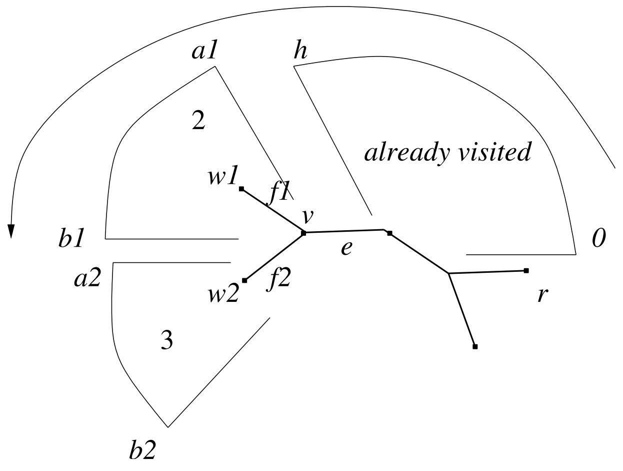
```
Set $a = h$ // number of nodes placed before recursion
Set $b = 0$ // number of nodes placed after recursion
**for all** edges $f \neq e$ adjacent to $v$ **do**
    Let $w \neq v$ be the other node adjacent to $f$
    Set $b = \text{setAngles}(a, f, w)$
    Set $\alpha(f) = \frac{\pi(a+b)}{n}$
    Set $a = b$
**return** $b$.
```

Lemma The algorithm computes the edge angles of a a circular embedding in linear time.

Proof Every edge is visited exactly once, hence the runtime is linear in the number of edges $|E|$ (which is linear in the number of leaves).

To see that the algorithm produces the correct result, consider the situation for some arbitrary edge e and node v. The parameter h corresponds to the highest index assigned so far. Let $f_1, \ldots, f_k$ be the list of edges adjacent to v that are considered by the algorithm, let w_i denote the corresponding other node, and let a_i and b_i be the values of a and b directly after processing f_i. The latter two numbers denote the range of indices recursively assigned to the set of leaves in the subtree rooted at w_i.

The situation when processing node v:



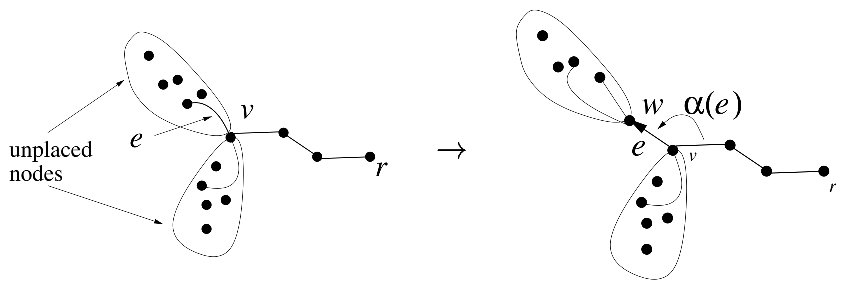
We then compute $\alpha(f_i) = \frac{\pi(a_i+b_i)}{n} = \frac{2\pi}{n} \cdot \frac{a_i+b_i}{2} = \frac{2\pi}{n} \cdot \frac{a_i+a_i+1+\ldots+b_i}{b_i-a_i+1}$, as required in the definition of a circular embedding. □

5.13 Determining coordinates

Given a phylogenetic tree T and an angle function $\alpha : E \to [0, 2\pi]$, how do we obtain coordinates for the nodes?

Idea: In a depth-first traversal of the tree starting at the reference node r, for every edge e, simply push the opposite node away from the current node in the direction specified by

$\alpha(e)$ by the amount specified by the length $\omega(e)$:



before processing e after processing e

The following algorithm does the pushing. Initially, we set $\epsilon(r) = (0,0)$ and invoke setCoordinates$(null, r)$.

```
**Algorithm** setCoordinates$(e, v)$
Input: Phylogenetic tree $T$ and edge angles $\alpha$
Output: Circular embedding $\epsilon$ of $T$
**begin**
Set $p = \epsilon(v)$
**for each** edge $f \neq e$ adjacent to $v$ **do**
    Let $w \neq v$ be the other node adjacent to $f$
    Obtain $p'$ by translating $p$ in direction $\alpha(e)$ by amount $\omega(e)$
    Set $\epsilon(w) = p'$
    **call** setCoordinates$(f, w)$
**end**
```

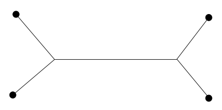
Theorem A circular embedding is computable in linear time.

Challenge: prove that the resulting configuration is indeed an embedding, that is, that no two edges can cross.

5.14 The number of edges and nodes of an unrooted phylogenetic tree

Let T be an unrooted phylogenetic tree on n taxa, i.e., with n leaves. How many nodes and edges does T have? Let us assume that T is binary. Any non-binary tree on n taxa will have less nodes and edges.

Consider a tree for $n = 4$, it has 6 nodes and 5 edges:

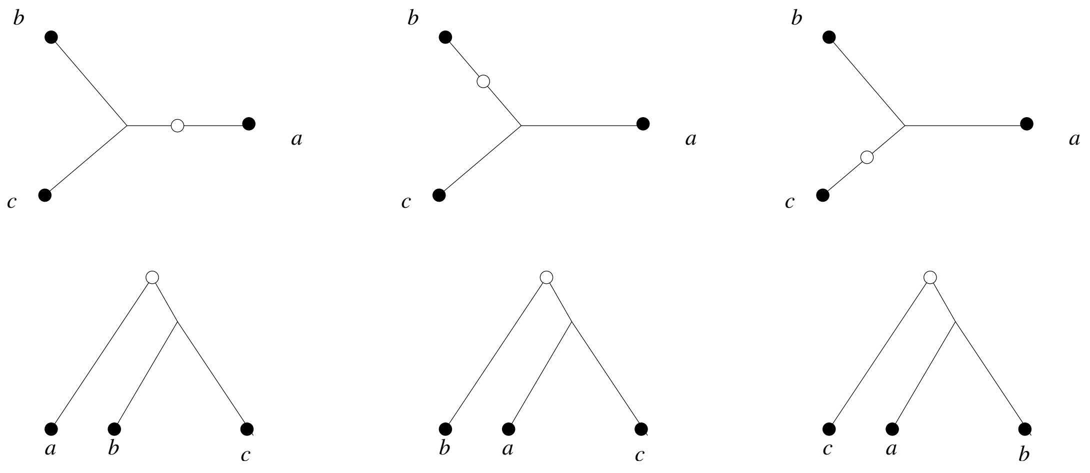


Now inductively, assume $n > 4$. Any tree T' with $n + 1$ leaves can be obtained from some tree T with n leaves by inserting a new node v into some edge e of T and connecting v to a new leaf w. This increases both the number of nodes and the number of edges by 2.

Putting this together, we see that the number of nodes is $2n-2$ and the number of edges is $2n-3$.

5.15 The number of phylogenetic trees

An unrooted tree T with n leaves has $2n-2$ nodes and $2n-3$ edges. A root can be added in any of the $2n-3$ edges, thus producing $2n-3$ different rooted trees from T:



For $n = 3$ there are three ways of adding a root. Similarly, there are 3 different ways of adding an extra edge with a new leaf to obtain an unrooted tree on 4 leaves. This new tree has $(2n-3) = 5$ edges and there are 5 ways to obtain a new tree with 5 leaves etc.

Continuing this, we see that there are

$$U(n) = (2n-5)!! := 3 \cdot 5 \cdot 7 \cdot \ldots \cdot (2n-5)$$

unrooted trees on n leaves. Similarly, there are

$$R(n) = (2n-3)!! = U(n) \cdot (2n-3) = 3 \cdot 5 \cdot \ldots \cdot (2n-3)$$

rooted trees.

These numbers grows very rapidly with n, for example, $U(10) \approx 2$ million and $U(20) \approx 2.2 \times 10^{20}$.

5.16 Constructing phylogenetic trees

There are three main approaches to constructing phylogenetic trees from molecular data.

1. Using a *distance method*, one first computes a distance matrix from a given set of biological data and then one computes a tree that represents these distances as closely as possible.

2. *Maximum parsimony* takes as input a set of aligned sequences and attempts to find a tree and a labeling of its internal nodes by auxiliary sequences such that the number of mutations along the tree is minimum.

3. Given a probabilistic model of evolution, *maximum likelihood* approaches aim at finding a phylogenetic tree that maximizes the likelihood of obtaining the given sequences.

5.17 Distances

Given a set $X = \{x_1, x_2, \ldots, x_n\}$ of taxa. The input to a distance method is a dissimilarity matrix $D : X \times X \to \mathbb{R}_{\geq 0}$ that associates a *distance* $d(x_i, x_j)$ with every pair of taxa $x_i, x_j \in X$. Sometimes we will abbreviate $d_{ij} := d(x_i, x_j)$ or $D_{ij} := d(x_i, x_j)$.

We usually require that

1. the matrix is *symmetric*, that is, $d(x, y) = d(y, x)$ for all $x, y \in X$, and

2. $d(x, x) = 0$ for all $x \in X$.

We call D a *pseudo metric*, if the *triangle inequalities* are satisfied:

$$d(x, z) \leq (x, y) + d(y, z) \text{ for all } x, y, z \in X,$$

and a *metric*, if additionally we have $d(x, y) > 0$ for all $x \neq y$.

5.18 Hamming distance

Let a collection of taxa be given by a set of distinct sequences $A = \{a_1, a_2, \ldots, a_n\}$ and assume we are given a multiple sequence alignment A^* of the sequences.

We define *sequence dissimilarity* as the (normalized) *Hamming distance* $Ham(a_i, a_j)$ between two taxa a_i and a_j as the number of mismatch positions in a_i^* and a_j^*, divided by the number of comparisons.

We ignore any column in which both sequences contain a gap. If only one sequence has a gap in a column then we can either ignore the column, or treat it as a match, or as a mismatch, depending on the type of data. Usually, one ignores *all* columns in which any of the n sequences contains the gap.

Lemma If the alignment is gap-less, then the corresponding distance matrix is a metric on A.

Proof: Consider three distinct sequences $a_i, a_j, a_k \in A$. If $a_i^* \neq a_k^*$, then either $a_i^* \neq a_j^*$, or $a_k^* \neq a_j^*$, and hence $Ham(a_i, a_k) < Ham(a_i, a_j) + Ham(a_j, a_k)$. $\square$

For protein data, it makes sense to relax the definition of *sequence dissimilarity* to the number of "non-synonymous" residues divided by the number of sequence positions compared.

For example, we may choose to ignore "conservative substitutions" by pooling amino acids with similar properties into six groups: acidic (D,E), aromatic (F,W,Y), basic (H,K,R), cysteine, non-polar (A,G,I,L,P,V), and polar (M,N,Q,S,T). Two residues are considered synonymous, if the are contained in the same group, and non-synonymous, otherwise.

Example:

$$\begin{array}{l}
a_1 \quad \texttt{C A A C C C C C A A A A A} \\
a_2 \quad \texttt{T A A T T T - C A A A A A} \\
a_3 \quad \texttt{C G G T T T - - A A A A A}
\end{array}$$

Distances:

$$Ham(a_1, a_2) = \frac{4}{12} = 0.\overline{33}$$

$$Ham(a_1, a_3) = \frac{5}{11} = 0.\overline{45}$$

$$Ham(a_2, a_3) = \frac{3}{11} = 0.\overline{27}$$

Hamming distances are only suitable for closely related sequences. Below we will discuss more sophisticated distances.

Question: What is the average Hamming distance between two random gap-free DNA sequences of the same length?

5.19 UPGMA

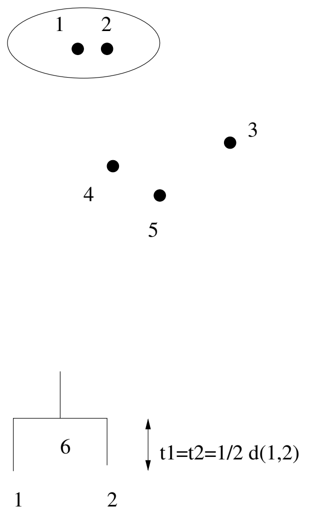
We will now discuss a simple distance method called UPGMA which stands for *unweighted pair group method using arithmetic averages* (Sokal & Michener 1958).

Given a set of taxa X and a distance matrix D, UPGMA produces a rooted phylogenetic tree T with edge lengths.

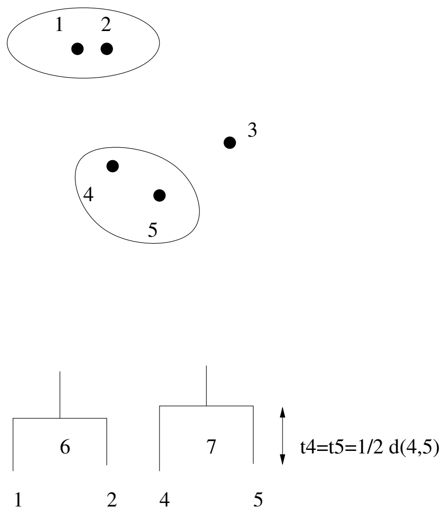
It operates by clustering the given taxa, at each stage merging two clusters and at the same time creating a new node in the tree. The tree is assembled "upwards", first clustering pairs of leaves, then pairs of clustered leaves etc. Each node is given a height and the edge lengths are obtained as the difference of heights of its two end nodes.

5.20 UPGMA example

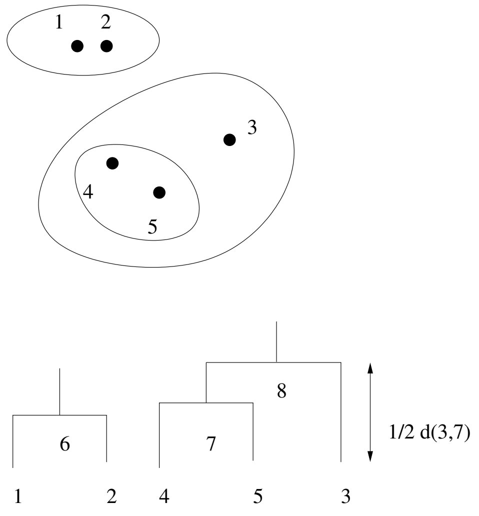
Example $X = \{1, 2, 3, 4, 5\}$, distances given by distance in the plane:



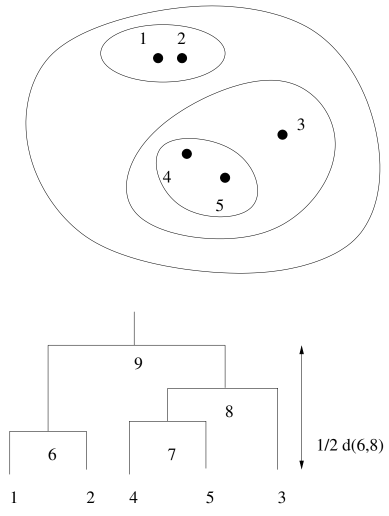
cluster 1 and 2



cluster 4 and 5



cluster 7 and 3



cluster 6 and 8

UPGMA produces a rooted, binary phylogenetic tree.

5.21 The distance between two clusters

Initially, we are given a distance $d(x, y)$ between any two taxa, i.e. leaves, x and y.

We define the distance $d(i, j) := d(C_i, C_j)$ between two clusters $C_i \subseteq X$ and $C_j \subseteq X$ to be the average distance between pairs of taxa from each cluster:

$$d(i, j) = \frac{1}{|C_i||C_j|} \sum_{x \in C_i, y \in C_j} d(x, y).$$

Note that, if C_k is the union of two clusters C_i and C_j, and C_l is any other cluster, then

$$d(k, l) = \frac{d(i, l)|C_i| + d(j, l)|C_j|}{|C_i| + |C_j|}.$$

This is a useful *update* formula, because using it in the algorithm, we can obtain the distance between two clusters in constant time.

5.22 The UPGMA algorithm

The UPGMA algorithm is very straight-forward:

```
Algorithm UPGMA

Input: A set of taxa $X$ and a corresponding distance matrix $D$
Output: A binary, rooted phylogenetic UPGMA tree on $T$

Initialization
    Assign each taxon $x_i$ to its own cluster $C_i$
    Define one leaf of $T$ for each taxon, placed at height zero

Iteration
    Determine two clusters $C_i$ and $C_j$ for which $d(i,j)$ is minimal
    Define a new cluster $k$ by $C_k = C_i \cup C_j$
    Define $d(k,l)$ for all existing clusters $l$ using the update formula
    Define a node $k$ with daughter nodes $i$ and $j$, and place it at height $\frac{d(i,j)}{2}$
    Add $C_k$ to the set of current clusters and remove $C_i$ and $C_j$.

Termination
    When only two clusters $C_i$ and $C_j$ remain, place the root at height $\frac{d(i,j)}{2}$.
```

(Problem: show that this algorithm produces well-defined edge lengths, i.e. a parent node always lies above its daughters.)

Finally, for each edge e, set the edge length $\omega(e)$ equal to the difference of the heights of the two incident nodes.

Example of UPGMA applied to 5S rRNA data:

Original distances:

	Bsu	Bst	Lvi	Amo	Mlu
Bsu	–	$\underline{0.1715}$	0.2147	0.3091	0.2326
Bst		–	0.2991	0.3399	0.2058
Lvi			–	0.2795	0.3943
Amo				–	0.4289
Mlu					

Abbreviations:
Bsu: *Bacillus subtilis*
Bst: *Baclillus stearothermophilus*
Lvi: *Lactobacillus viridescens*
Amo: *Acholeplasma modicum*
Mlu: *Micrococcus luteus*

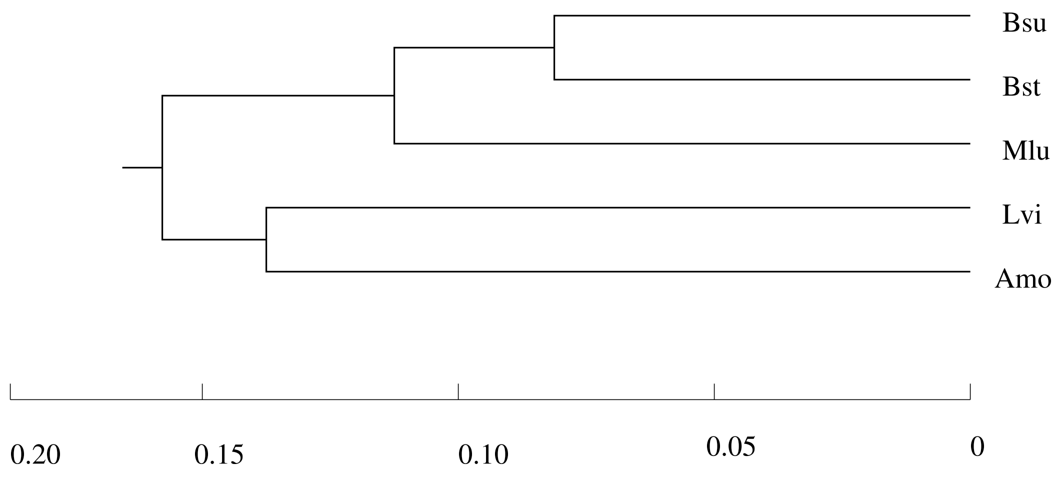
$\rightarrow$

	Bsu + Bst	Lvi	Amo	Mlu
Bsu + Bst	–	0.2569	0.3245	$\underline{0.2192}$
Lvi		–	0.2795	0.3943
Amo			–	0.4289
Mlu				–

$$\rightarrow \quad \begin{array}{r|ccc} & \text{Bsu} + \text{Bst} + \text{Mlu} & \text{Lvi} & \text{Amo} \\ \text{Bsu} + \text{Bst} + \text{Mlu} & - & 0.3027 & 0.3593 \\ \text{Lvi} & & - & \underline{0.2795} \\ \text{Amo} & & & - \end{array}$$

$$\rightarrow \quad \begin{array}{r|cc} & \text{Bsu} + \text{Bst} + \text{Mlu} & \text{Lvi} + \text{Amo} \\ \text{Bsu} + \text{Bst} + \text{Mlu} & - & \underline{0.3310} \\ \text{Lvi} + \text{Amo} & & - \end{array}$$

The resulting tree:



This tree is biologically incorrect, as we will see later.

5.23 Tree reconstruction

The goal of phylogenetic analysis is usually to *reconstruct* a phylogenetic tree from data, such as distances or sequences, that was produced by some *generating* or *true* tree.

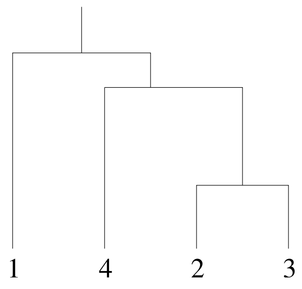
When reconstructing trees from real data, the generating tree is the path of events that evolution actually took. In this case, the true tree is unknown and the objective is, of course, to try and reconstruct it.

In contrast, in simulation studies, a *known* tree T_0 is used to generate artificial sequences and/or distances, under some specified model of evolution. A tree reconstruction method is then applied and its performance can be evaluated by comparing the resulting tree T with the true tree T_0.

Definition Given a phylogenetic tree T. We say that a distance matrix D is *directly obtainable* from T, if it was obtained by adding up edge lengths on paths between leaves.

5.24 The molecular clock hypothesis

Given a distance matrix D, the UPGMA method aims at building a rooted tree T with the property that all leaves have the same distance from the root ρ:

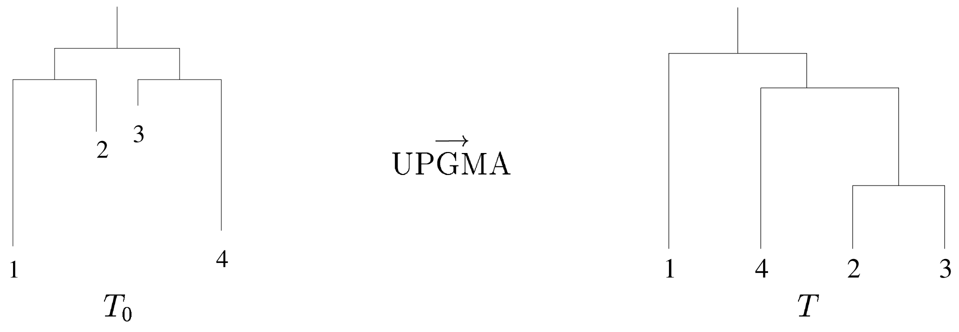


This approach is suitable for sequence data that has evolved under circumstances in which the rate of mutations of sequences is constant over time and for all lineages in the tree.

Definition The assumption that evolutionary events happen at a constant rate is called the *molecular clock* hypothesis.

5.25 UPGMA and the molecular clock

If the input distance matrix D was directly obtained from a generating phylogenetic tree T_0 that adheres to the molecular clock assumption, then the tree T reconstructed by UPGMA from D will equal T_0. Otherwise, if T_0 does not do so, then UPGMA may fail to reconstruct the tree correctly, for example:



The problem here is that the closest leaves in T_0 are not neighboring leaves: they do not have a common parent node.

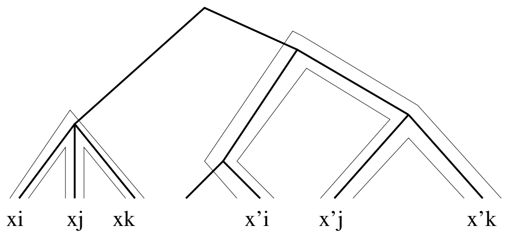
5.26 The ultrametric property

A distance matrix D is called an *ultrametric*, if for any triplet of taxa $x_i, x_j, x_k \in X$, the three distances $d(x_i, x_j)$, $d(x_i, x_k)$ and $d(x_j, x_k)$ have the property that either:

1. all three distances are equal, or

2. two are equal and the remaining one is smaller.

Note that if D was directly obtained from some tree T that satisfies the molecular clock hypothesis, then D is an ultrametric:



condition (1) condition (2)

We say that a rooted phylogenetic tree T is *ultrametric*, if every leaf has the same distance from the root.

One can show the following result:

Theorem A distance matrix D is directly obtainable from some ultrametric tree T, if and only if D is ultrametric.

5.27 Estimating the deviation from a molecular clock

Given a distance matrix D obtained by comparison of sequences generated along some unknown tree T_0. Biologically, it may be of interest to know how well the molecular clock hypothesis holds. In other words, how close is D to being an ultrametric?

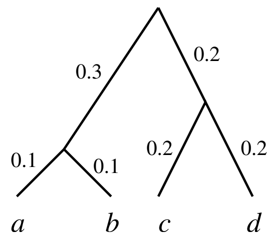
To answer this question, we define the *stretch of an ultrametric* U with respect to D as follows:

$$stretch_D(U) = \max\left\{\frac{D_{ij}}{U_{ij}}, \frac{U_{ij}}{D_{ij}} \mid i,j \in X\right\}.$$

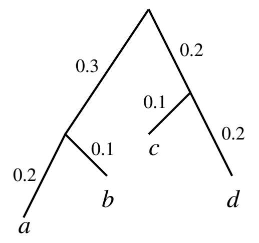
The *stretch of* D is defined as the minimum stretch over all possible ultrametrics: $stretch(D) = \min_U\{stretch_D(U)\}$ and gives a lower bound for the stretch of any tree obtained from D.

This value can be computed in $O(n^2)$ time, see L. Nakhleh, U. Roshan, L. Vawter and T. Warnow, LNCS 2452:287-299 (2002).

Example of an ultrametric U and a non-ultrametric D:



$$\rightarrow \quad U = \left\{ \begin{array}{ccccc} & a & b & c & d \\ a & - & 0.2 & 0.8 & 0.8 \\ b & & - & 0.8 & 0.8 \\ c & & & - & 0.4 \\ d & & & & - \end{array} \right.$$



$$\rightarrow \quad D = \left\{ \begin{array}{ccccc} & a & b & c & d \\ a & - & 0.3 & 0.8 & 0.9 \\ b & & - & 0.7 & 0.8 \\ c & & & - & 0.3 \\ d & & & & - \end{array} \right.$$

The stretch of U w.r.t. D is:

$$stretch_D(U) = \max\left\{\frac{0.3}{0.2}, \frac{0.9}{0.8}, \frac{0.8}{0.7}, \frac{0.4}{0.3}\right\} = \frac{3}{2}.$$

5.28 Additivity and the four-point condition

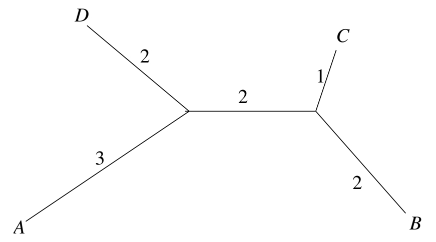
Given a set of taxa X. A distance matrix D on X is called *additive*, if D is directly obtainable from some phylogenetic tree T.

Given an arbitrary distance matrix D. Can we determine whether D is additive without attempting to compute an appropriate tree? The answer is yes, using the following result due to Peter Buneman (1971):

Theorem A distance matrix D on X is additive, iff for any four (not necessarily distinct) taxa $x_i, x_j, x_k, x_l \in X$ the so-called *four-point condition* holds:

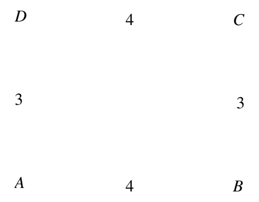
$$d(x_i, x_j) + d(x_k, x_l) \leq \max\left(d(x_i, x_k) + d(x_j, x_l), d(x_i, x_l) + d(x_j, x_k)\right).$$

Distances obtained directly from a phylogenetic tree:



Check the four-point condition with:
$d(A,B) + d(C,D) = 7 + 5 = 12$
$d(A,C) + d(B,D) = 6 + 6 = 12$
$d(A,D) + d(B,C) = 5 + 3 = 8$
$\Rightarrow$ the four-point condition holds.

Euclidean distances in the plane:



Check the four-point condition with:
$d(A,B)+d(C,D)=4+4=8$
$d(A,C)+d(B,D)=5+5=10$
$d(A,D)+d(B,C)=3+3=6$
$\Rightarrow d(A,C)+d(B,D)\not\leq$
$\max\left(d(A,B)+d(C,D), d(A,D)+d(B,C)\right)$
$\Rightarrow$ 4-point condition doesn't hold.

5.29 Neighbor-joining on a known tree

The most widely used distance method is the *neighbor-joining (NJ)* method, originally introduced by Saitou and Nei (1987). Given a distance matrix D, neighbor-joining produces an unrooted phylogenetic tree T with edge lengths. It is especially suitable, when the rate of evolution of the separate lineages under consideration varies.

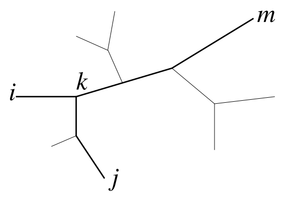
First, consider a tree T and let D be the distance matrix directly obtainable from T, i.e. D is a distance matrix defined on the leaves of T, obtained by adding edge lengths. In preparation of introducing the neighbor-joining algorithm, let us first understand how one could reconstruct T from D.

The following step reduces the number of leaves by one and we can repeatedly apply it until we arrive at a single pair of leaves:

Find a pair of *neighboring leaves*, i.e. leaves that have the same parent node, k. Suppose their numbers are i,j. Remove i,j from the list of nodes and add k to the current list of nodes, defining its distance to leaf m by

$$d_{km}=\frac{1}{2}(d_{im}+d_{jm}-d_{ij}).$$

By additivity of D, the distances d_{km} defined in this way are precisely those between equivalent nodes in the original tree:



For any three leaves i,j,m there is a node k where the paths to them meet. By additivity,

$$d_{im}=d_{ik}+d_{km},\ \ d_{jm}=d_{jk}+d_{km} \text{ and } d_{ij}=d_{ik}+d_{jk},$$

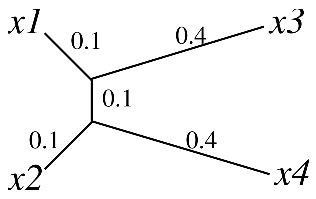
which implies $d_{km}=\frac{1}{2}(d_{im}+d_{jm}-d_{ij})$.

5.30 Neighbor-joining

In the above discussion, we assumed that the tree T is known and used it to determine which leaves are neighbors.

The neighbor-joining method is based on the fact that we can decide which nodes are neighboring *without* knowing the tree, but only using the distance matrix.

However, it does *not* suffice simply to pick the two closest leaves, i.e. a pair i, j with d_{ij} minimal, for example:



Given distances generated by this tree. Leaves x_1 and x_2 have minimal distance, but are not neighbors.

To avoid this problem, the trick is to subtract the averaged distances to all other leaves, thus compensating for long edges. We define:

$$N_{ij} := d_{ij} - (r_i + r_j),$$

where

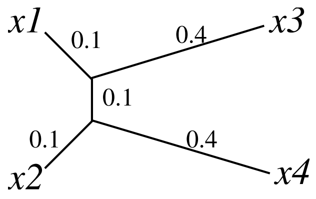
$$r_i = \frac{1}{|L| - 2} \sum_{k \in L} d_{ik},$$

and L denotes the set of leaves.

Theorem If D is directly obtainable from some tree T, then the two leaves x_i and x_i for which N_{ij} is minimal are neighbors in T.

This result ensures that the neighbor-joining algorithm will correctly reconstruct a tree from its additive distances.

Let us illustrate this result using the previous example:



Here, $r_1 = 0.7$, $r_2 = 0.7$, $r_3 = 1.0$ and $r_4 = 1.0$. And so,

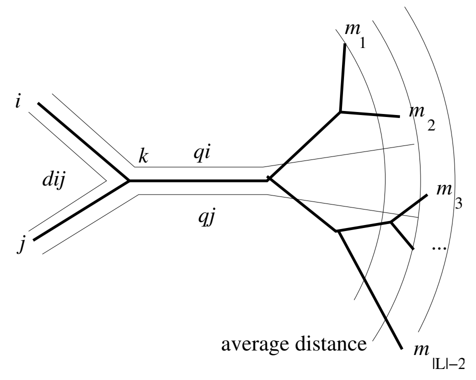
$$N = \left\{ \begin{array}{lcccc} & x_1 & x_2 & x_3 & x_4 \\ x_1 & - & -1.1 & -\mathbf{1.2} & -1.1 \\ x_2 & & - & -1.1 & -\mathbf{1.2} \\ x_3 & & & - & -1.1 \\ x_4 & & & & - \end{array} \right.$$

The matrix N attains a minimum value for the pair $i = 1$ and $j = 3$ and for the pair $i = 2$ and $j = 4$, as required.

5.31 The neighbor-joining algorithm

```
**Algorithm** (Neighbor-joining)
Input: Distance matrix $D$
Output: Phylogenetic tree $T$
Initialization:
        Define $T$ to be the set of leaf nodes, one for each taxon.
        Set $L = T$.
Iteration:
        Pick a pair $i, j \in L$ for which $N_{ij}$ is minimal.
        Define a new node $k$ and
            set $d_{km} = \frac{1}{2}(d_{im} + d_{jm} - d_{ij})$, for all $m \in L$.
        Add $k$ to $T$ with edges of lengths $d_{ik} = \frac{1}{2}(d_{ij} + r_i - r_j)$ and
            $d_{jk} = d_{ij} - d_{ik}$, joining $k$ to $i$ and $j$, respectively.
        Remove $i$ and $j$ from $L$ and add $k$.
Termination:
        When $L$ consists of two leaves $i$ and $j$, add the remaining
            edge between $i$ and $j$, with length $d_{ij}$.
```

Why do we use $d_{ik} = \frac{1}{2}(d_{ij}+r_i-r_j)$ to update distances? By definition, $r_i = \frac{1}{|L|-2}\sum_{m\in L} d_{im} = \frac{1}{|L|-2}\sum_{m\neq i,j} d_{im} + \frac{d_{ij}}{|L|-2}$. In other words, r_i is the average distance $q_i = \frac{1}{|L|-2}\sum_{m\neq i,j} d_{im}$ to all other nodes $m \neq i, j$, plus $\frac{d_{ij}}{|L|-2}$, as indicated here:



As we see from this figure: $2d_{ik} = d_{ij} + q_i - q_j$. This is equivalent to the update formula.

5.32 Application of neighbor-joining

Given an additive distance matrix D directly obtained from a phylogenetic tree T, neighbor-joining is guaranteed to reconstruct T correctly.

However, in practice we are never given a matrix that was "directly obtained" from the generating tree, but rather the distance matrix is usually obtained very indirectly by a comparison of finite sequence data generated along the tree. Such data is rarely additive. Nevertheless, the neighboring-joining method is often applied to such data and has proved to be a fast, useful and robust tree reconstruction method.

5.33 Example

Original data:

$$D_0 = \left\{ \begin{array}{ccccc|c} & x_1 & x_2 & x_3 & x_4 & r \\ x_1 & - & 3 & 5 & 6 & 7 \\ x_2 & 3 & - & 6 & 5 & 7 \\ x_3 & 5 & 6 & - & 9 & 10 \\ x_4 & 6 & 5 & 9 & - & 10 \end{array} \right. \quad \rightarrow \quad N_0 = \left\{ \begin{array}{ccccc} & x_1 & x_2 & x_3 & x_4 \\ x_1 & - & -11 & -12 & -11 \\ x_2 & & - & -11 & -12 \\ x_3 & & & - & -11 \\ x_4 & & & & - \end{array} \right.$$

Data after one merge of neighbors:

$$D_1 = \left\{ \begin{array}{cccc|c} & x_1+x_3 & x_2 & x_4 & r \\ x_1+x_3 & - & 2 & 5 & 7 \\ x_2 & & - & 5 & 7 \\ x_4 & & & - & 10 \end{array} \right. \quad \rightarrow \quad N_1 = \left\{ \begin{array}{cccc} & x_1+x_3 & x_2 & x_4 \\ x_1+x_3 & - & -12 & -12 \\ x_2 & & - & -12 \\ x_4 & & & - \end{array} \right.$$

Data after two merges of neighbors:

$$D_2 = \left\{ \begin{array}{ccc|c} & x_1+x_3 & x_2+x_4 & r \\ x_1+x_3 & - & 1 & \\ x_2+x_4 & & - & \end{array} \right. \quad \rightarrow \quad N_2 = \left\{ \begin{array}{ccc} & x_1+x_3 & x_2+x_4 \\ x_1+x_3 & - & quad \\ x_2+x_4 & & - \end{array} \right.$$

5.34 Example

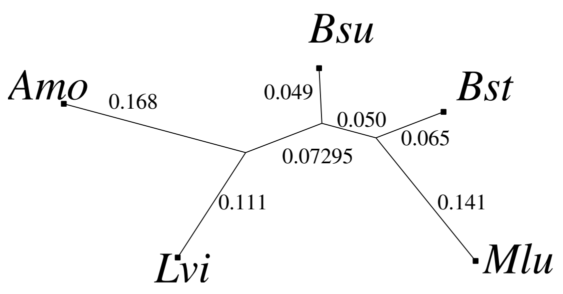
Example of neighbor-joining applied to 5S rRNA data:

Original distances:

Abbreviations:
Bsu: *Bacillus subtilis*
Bst: *Baclillus stearothermophilus*
Lvi: *Lactobacillus viridescens*
Amo: *Acholeplasma modicum*
Mlu: *Micrococcus luteus*

	Bsu	Bst	Lvi	Amo	Mlu
Bsu	–	0.1715	0.2147	0.3091	0.2326
Bst		–	0.2991	0.3399	0.2058
Lvi			–	0.2795	0.3943
Amo				–	0.4289
Mlu					

The resulting tree:



5.35 Rooting unrooted trees

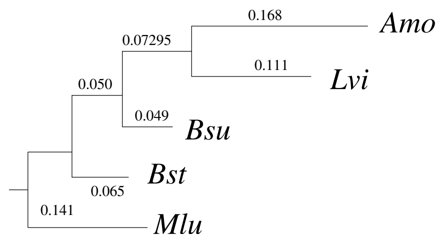
In contrast to UPGMA, most tree reconstruction methods produce an *unrooted* tree. Indeed, determining the root of a tree using computational methods is very difficult.

In practice, the question of rooting a tree is addressed by adding an *outgroup* to the set of taxa under consideration. This is a taxon that is more distantly related to all other taxa then any other of the taxa.

The root is then assumed to be on the branch attaching the outgroup taxon to the rest of the tree.

In practice, selecting an appropriate outgroup can be difficult: if it is too similar to the other taxa, then it might be more related to some than to others. If it is too distant, then there might not be enough similarity to the other taxa to perform meaningful comparisons.

In the above neighboring tree, Mlu is the outgroup, Hence, the rooted version of this tree looks something like this:



This tree is believed to be closer to the correct tree than the one produced earlier using UPGMA. In particular, the UPGMA tree does not separate the outgroup from all other taxa by the root node. The reason why UPGMA produces an incorrect tree is that two of the sequences, those of *L.viridescens* (Lvi) and *A.modicum* (Amo) are very much more diverged than the others.

5.36 Models of evolution

In phylogenetic analysis, a *model of evolution* is given by a rooted tree T, called the *model-*, *true-* or *generating* tree, together with a procedure for generating sequences along the model tree.

Usually, the procedure must determine how to generate an initial sequence at the root of the tree and specify how to "evolve" sequences along the edges of the tree. This involves obtaining intermediate sequences for all internal nodes of the tree, and producing a set of aligned sequences A at the leaves of the tree.

5.37 The Jukes-Cantor model of evolution

T. Jukes and C. Cantor (1969) introduced a very simple model of DNA sequence evolution:

Definition Let T be a rooted phylogenetic tree. The *Jukes-Cantor* model of evolution makes the following assumptions:

1. The possible states for each site are `A`, `C`, `G` and `T`.
2. The initial sequence length is an input parameter and for each site the state at the root is drawn from a given distribution (typically uniform).
3. The sites evolve identically and independently (i.i.d.) down the edges of the tree from the root at a fixed rate u.
4. With each edge $e \in E$ we associate a duration $t = t(e)$ and the expected number of mutations per site along e is $u\tau(e)$. The probabilities of change to each of the 3 remaining states are equal.

How do we "evolve" a sequence down an edge e under the Jukes-Cantor model?

Let $v = (v_1, \ldots, v_n)$ and $w = (w_1, \ldots, w_n)$ denote the source and target sequences associated with e. We assume that v has already been determined and we want to determine w.

Under the Jukes-Cantor model, the evolutionary event $C_3 =$ *nucleotide changes to one of the other three bases* occurs at a fixed rate of u.

Now consider the event $C_4 =$ *nucleotide changes to any base.* The rate of occurrences of this event is taken to be $\frac{4}{3}u = 4\frac{u}{3}$.

The probability that event C_4 *does not* occur within time t is: $e^{-\frac{4}{3}ut}$ (Poisson distribution with $k = 0$).

Given that the event C_4 occurs, the probability of ending in any particular base is $\frac{1}{4}$.

Let P and Q denote the base pair present at a given site before and after a given time period t. The probability that an event of type C_4 occurs and changes $P \in \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$ to base Q within time t is:

$$\text{Prob}(Q \mid P, t) = \frac{1}{4}\left(1 - e^{-\frac{4}{3}ut}\right).$$

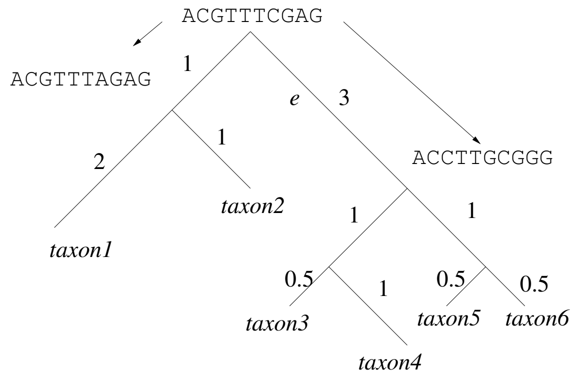
Now, adding the probability that the event C_4 does not occur, we obtain the probability that the state does not change, i.e. $P = Q$:

$$\text{Prob}(Q = P \mid P, t) = e^{-\frac{4}{3}ut} + \frac{1}{4}\left(1 - e^{-\frac{4}{3}ut}\right) = \frac{1}{4}\left(1 + 3e^{-\frac{4}{3}ut}\right).$$

Thus, we obtain a *probability-of-change formula* for the probability of an *observable* change occurring at any given site in time t:

$$\begin{aligned}\text{Prob}(\text{change} \mid t) &= 1 - \text{Prob}(Q = P \mid P, t) = 1 - \tfrac{1}{4}\left(1 + 3e^{-\frac{4}{3}ut}\right) \\ &= \tfrac{3}{4}\left(1 - e^{-\frac{4}{3}ut}\right).\end{aligned}$$

We can use this model to generate artificial sequences along a model tree. E.g., set the sequence length to 10 and the mutation rate to $u = 0.1$. Initially, the root node is assigned a random sequence. Then the sequences are evolved down the tree, at each edge using the probability-of-change formula to decide whether a given base is to change:



The probability of change along edge e is $0.75(1 - e^{-\frac{4}{3}\times 0.1\times 3}) = 0.75(1 - e^{-0.4}) = 0.247$.

5.38 The Jukes-Cantor distance transformation

Given sequences generated under the Jukes-Cantor model of evolution. We would like to calculate the expected number of mutations for any given site on the path in a Jukes-Cantor tree between the leaves a_i and a_j.

Lemma The maximum likelihood distance between a pair of sequences a_i, a_j (that is, the most likely ut to have generated the observed sequences) is given by the following formula:

$$JC(a_i, a_j) = -\frac{3}{4}\ln\left(1 - \frac{4}{3}Ham(a_i, a_j)\right).$$

This is called the *Jukes-Cantor distance transformation.*

Proof Asymptotically, for longer and longer sequences, $Ham(a_i, a_j)$ converges to the probability that any given site will have a different value at a_i than at a_j.

Based on this, we obtain the Jukes-Cantor estimation by setting:

$$Ham(a_i, a_j) = \frac{3}{4}\left(1 - e^{-\frac{4}{3}ut}\right),$$

thus,

$$e^{-\frac{4}{3}ut} = 1 - \frac{4}{3}Ham(a_i, a_j),$$

and so:

$$ut = -\frac{3}{4}\ln\left(1 - \frac{4}{3}Ham(a_i, a_j)\right).$$

$\square$

5.39 Accounting for superimposed events

Hamming distances are suitable for inferring phylogenies between closely related species, given long sequences. For more distantly related sequences, the problem arises that mutation events will take place more than once at the same site. These *superimposed events* will not contribute to the Hamming distances and thus will go undetected.

Note that the expected Hamming distance between two random sequences is $\frac{3}{4}$. Any two sequences a_i, a_j for which $Ham(a_i, a_j) \geq \frac{3}{4}$ holds are called *saturated* w.r.t. each other.

Note that the Jukes-Cantor transformation is undefined for any pair of saturated sequences. In practice, when a saturated pair of taxa is encountered, their JC value is simply set to a large number which may be a fixed-factor times the largest value obtained between any two non-saturated taxa, for example.

5.40 More general transformations

The Jukes-Cantor model assumes that all nucleotides occur with the same frequency. This assumption is relaxed under the *Felsenstein 81* model introduced by Joe Felsenstein (1981), which has the following transformation:

$$F81(a_i, a_j) = -B\ln(1 - Ham(a_i, a_j)/B),$$

where $B = 1 - (\pi_{\mathtt{A}}^2 + \pi_{\mathtt{C}}^2 + \pi_{\mathtt{G}}^2 + \pi_{\mathtt{T}}^2)$ and π_c is the frequency of base c. The base frequency is obtained from the pair of sequences to be compared, or better, from the complete set of given sequences.

Note that this contains the Jukes-Cantor transformation as a special case with $B = \frac{3}{4}$.

Unlike the Jukes-Cantor transformation, this transformation can also be applied to protein sequences, setting $B = \frac{19}{20}$ if all amino acids are generated with the same frequency, and $B = \sum_{i=1}^{20} \pi_i^2$, in the proportional case.

Both of these transformations assume that all changes of states are equally likely:

$$\begin{tikzcd}
\mathrm{A} \arrow[r, leftrightarrow, "u"] \arrow[d, leftrightarrow, "u"'] \arrow[dr, leftrightarrow] & \mathrm{C} \arrow[d, leftrightarrow, "u"] \arrow[dl, leftrightarrow, "u"] \\
\mathrm{G} \arrow[r, leftrightarrow, "u"] & \mathrm{T}
\end{tikzcd}$$

However, this is a very unrealistic assumption. For example, in DNA sequences, we observe many more *transitions*, which are purine-to-purine or pyramindine-to-pyramindine substitutions, than *transversions*, which change the type of the nucleotide. We need to model two separate substitution rates α and β:

$$\begin{tikzcd}
A \arrow[r, leftrightarrow, "\alpha"] \arrow[d, leftrightarrow, "\beta"] \arrow[dr, leftrightarrow, "\alpha"] & C \arrow[d, leftrightarrow, "\beta"] \arrow[dl, leftrightarrow, "\alpha"'] \\
G \arrow[r, leftrightarrow, "\alpha"] & T
\end{tikzcd}$$

(Transitions: `A` $\leftrightarrow$ `G`, `C` $\leftrightarrow$ `T`, transversions: `A` $\leftrightarrow$ `T`, `G` $\leftrightarrow$ `T`, `A` $\leftrightarrow$ `C`, and `C` $\leftrightarrow$ `G`.)

Given equal base frequencies, but different proportions P and Q of transitions and transversions between a_i and a_j, the distance for the *Kimura 2 parameter* model is computed as:

$$K2P(a_i, a_j) = \frac{1}{2}\ln\left(\frac{1}{1-2P-Q}\right) + \frac{1}{4}\ln\left(\frac{1}{1-2Q}\right).$$

If we drop the assumption of equal base frequencies, then we obtain the *Felsenstein 84* transformation:

$$F84(a_i, a_j) = -2A\ln\left(1 - \frac{P}{2A} - \frac{(A-B)Q}{2AC}\right) + 2(A-B-C)\ln\left(1 - \frac{Q}{2C}\right),$$

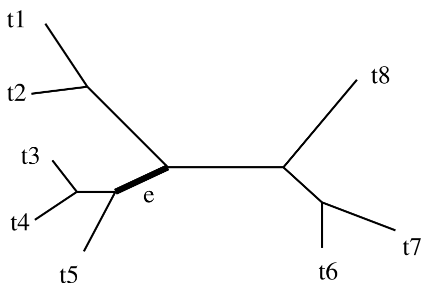
where $\pi_Y = \pi_C + \pi_T$, $\pi_R = \pi_A + \pi_G$, $A = \pi_C\pi_T/\pi_Y + \pi_A\pi_G/\pi_R$, $B = \pi_C\pi_T + \pi_A\pi_G$, and $C = \pi_R\pi_Y$.

Even more general models exist, but we will skip them...

5.41 Trees and splits

Any edge e of T defines a *split* $S = \{A, \bar{A}\}$ of X, that is, a partitioning of X into two non-empty sets A and $\bar{A}$, consisting of all taxa on the one side and other side of e, respectively.

For example:



Here, $A = \{t_3, t_4, t_5\}$ and $\bar{A} = \{t_1, t_2, t_6, t_7, t_8\}$.

We will use $\Sigma(T)$ to denote the *split encoding of* T, i.e. the set of all splits obtained from T. If $x \in X$ and $S \in \Sigma$, then we use $S(x)$ or $\bar{S}(x)$ to denote the split part that contains x, or doesn't contain x, respectively. We define the *size* of a split $S = \{A, \bar{A}\}$ as $\text{size}(S) = \min(|A|, |\bar{A}|)$. A split of size 1 is called a *trivial* split.

5.42 Compatible splits

Given a set of taxa X. Let Σ be a set of splits of X. Two splits $S_1 = \{A_1, \bar{A}_1\}$ and $S_2 = \{A_2, \bar{A}_2\}$ are called *compatible*, if one of the four following intersections

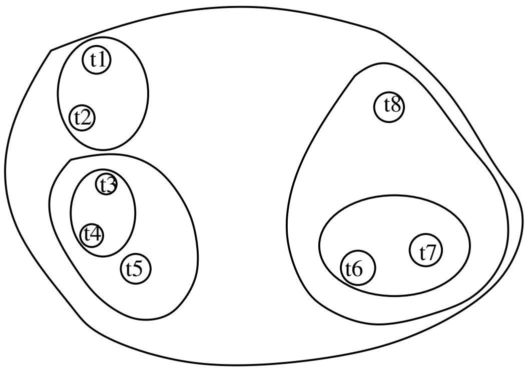
$$A_1 \cap A_2, \quad A_1 \cap \bar{A}_2, \quad \bar{A}_1 \cap A_2, \quad \text{or} \quad \bar{A}_1 \cap \bar{A}_2,$$

is empty.

A set Σ of splits of X is called *compatible*, if every pair of splits in Σ is compatible.

Example
Given the taxa set $X = \{a, b, c, d, e\}$. The splits $S_1 = \{\{a, b\}, \{c, d, e\}\}$, $S_2 = \{\{a, b, c\}, \{d, e\}\}$ and $S_3 = \{\{e\}, \{a, b, c, d\}\}$ are all compatible with each other. However, $S_4 = \{\{a, c\}, \{b, d, e\}\}$ is not compatible with the first one. Hence, the set $\Sigma = \{S_1, S_2, S_3\}$ is compatible, but $\Sigma' = \{S_1, S_2, S_3, S_4\}$ is not.

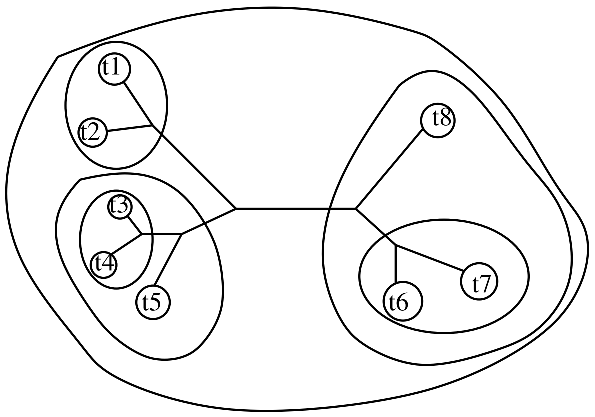
The compatibility condition states that any split S subdivides either the one side, or the other side, of any other split S', but not both sides. Hence, any set of compatible splits can be drawn as follows, without crossing lines:



This figure also shows the relationship between compatible splits and a *hierarchical clustering.* (A *hierarchical clustering* of a set X is a system $\mathcal{H}$ of subsets of X such that $\bigcup_{A \in \mathcal{H}} A = X$ and $A, B \in \mathcal{H} \Rightarrow$ either $A \cap B = \emptyset$, or $A \subset B$ or $B \subset A$.)

5.43 Compatible splits and trees

Any compatible set of splits Σ gives rise to a phylogenetic tree T, for example:



Note: To obtain a one-to-one correspondence between trees and compatible split systems, we now use the relaxed definition of a phylogenetic tree that allows any leaf to carry more than one label and also allows labeling of internal nodes, too.

Vice versa, any tree T gives rise to a compatible set of splits.

(Proof: Consider two edges e and e'. Because T is a tree, e and e' are connected by a unique path P:

$$v \xleftrightarrow{e} w \longleftrightarrow P \longleftrightarrow w' \xleftrightarrow{e'} v'$$

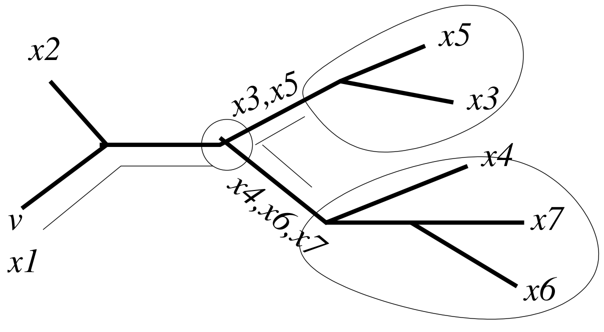
Because T is a tree, the set A, consisting of all nodes reachable from node v not using edge e, is disjoint from A', the set of all nodes reachable from node v' not using edge e'. Hence, the corresponding splits $S = \{A, \bar{A} = X \setminus A\}$ and $S' = \{A', \bar{A}' = X \setminus A'\}$ are compatible.)

In summary:

Theorem A set of splits Σ is compatible, iff there exists a phylogenetic tree T such that $\Sigma = \Sigma(T)$.

5.44 Splits from a tree

Given an unrooted phylogenetic tree T on $X = \{x_1, \ldots, x_n\}$. We can easily compute the set $\Sigma(T)$ in $O(n)$, where n is the number of taxa:



Starting at the leaf labeled x_1, visit all nodes in a post-order traversal, for each edge maintaining and reporting the set of leaves reached after crossing the edge.

The following algorithm recursively visits all nodes and prints out a split after visiting all nodes reachable over a particular edge. Initially, the algorithm is called with $e = null$ and v equal to the node labeled x_1.

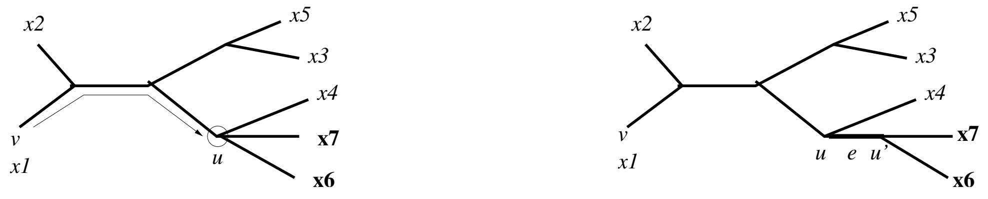
```
Algorithm TreeToSplits($e, v$)
Input: A phylogenetic tree $T$ on $X$
Output: The corresponding compatible splits $\Sigma(T)$
Returns: The set $\lambda(e)$ of all taxa separated from $x_1$ by $e$

begin Set $\lambda(e) = \emptyset$
for each edge $f \neq e$ adjacent to $v$ do
    Let $w$ be the opposite node adjacent to $f$
    call TreeToSplits($f, w$) and add the returned labels to $\lambda(e)$
print the split $\{\lambda(e), \overline{\lambda(e)}\}$
return $\lambda(e)$
```

5.45 A tree from splits

Given a compatible set of splits $\Sigma = \{S_1, S_2, \ldots, S_m\}$ of X.

Assume we have already processed the first i splits and have obtained a tree T_i. To incorporate a new edge e to represent the next split $S_{i+1} \in \Sigma$, starting at the node v labeled x_1, we follow a path through the tree until we reach the node w at which e is to be inserted:

Example: insert split $\{x_1, x_2, x_3, x_4, x_5\}$ vs $\{x_6, x_7\}$:



This node is found as follows: if there is only one edge leaving the current node u that *separates* x_1 from elements in $\bar{S}(x_1)$, then we follow this edge. If more than one separating edges exist, then $u = w$ and we create a new edge e from u to a new node u' and all separating edges are moved from u to u'.

```
Algorithm (Splits to tree)
Input: A set $\Sigma = \{S_1, \ldots, S_m\}$ of compatible splits of $X$,
(including all trivial ones)
Output: The corresponding phylogenetic tree $T = (V, E)$ on $X$
Initialization: Let $T$ be a star tree with $n$ leaves labeled $x_1, \ldots, x_n$
Orient all edges away from the node $v$ labeled $x_1$
For $e \in E$, maintain the set $\tau(e)$ of all taxa separated from $x_1$ by $e$
```

```
begin
for each non-trivial split $S \in \Sigma$ do
    Set $u = v$
    while there is only edge $e$ leaving $u$ with $\tau(e) \cap \bar{S}(x_1) \neq \emptyset$ do
        Set $u$ equal to the opposite node of $e$
    Let $F$ be the set of all edges $f$ leaving $u$ with $\tau(f) \cap \bar{S}(x_1) \neq \emptyset$
    Create a new edge $e$ from $u$ to a new node $u'$
    Set $\tau(e) = \bigcup_{f \in F} \tau(f)$
    Make all edges in $F$ leave from $w'$ instead of $w$
end
```

Lemma The algorithm constructs the correct tree T in $O(n \log n)$ expected steps.

Proof Assume that we have correctly processed splits $S_1 \ldots, S_p$ and that $\tau(e_i) = \bar{S}_i(x_1)$ for all $i \leq p$. By compatibility of Σ, for every edge e_i we must have either

$$\bar{S}_{p+1}(x_1) \subsetneq \bar{S}_i(x_1) \text{ or } \bar{S}_i(x_1) \subsetneq \bar{S}_{p+1}(x_1).$$

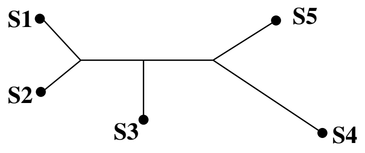
In the former case, the new edge e_{p+1} for S_{p+1} must be inserted on the other side of e_i and this is ensured by the while loop. In the latter case, the while loop is terminated and the edge e_{p+1} is placed between x_1 and all such edges e_i, as required.

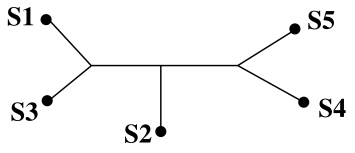
There are $O(n)$ splits and the average path distance between any edge and leaf is $\log n$. In summary, the algorithm takes $O(n \log n)$ steps. □

5.46 Comparison of two trees

Given two unrooted phylogenetic trees T_1 and T_2 on the same set of taxa X. We will call the first tree T_1 the *model* tree and T_2 the *reconstructed* tree. How can we measure how similar their topologies are?

Let $\Sigma_1 = \Sigma(T_1)$ and $\Sigma_2 = \Sigma(T_2)$ be the split encodings of the two trees. We define the set of all *false positives (FP)* as $\Sigma_2 \setminus \Sigma_1$. These are all splits contained in the estimated tree that are not present in the model tree. Similarly, we define the set of *false negatives (FN)* as $\Sigma_1 \setminus \Sigma_2$ as the set of all splits missing in the estimated tree, that are present in the model tree.





In this example, there is one false positive $\{S_1, S_3\}$ vs $\{S_2, S_4, S_5\}$ and one false negative $\{S_1, S_2\}$ vs $\{S_3, S_4, S_5\}$.

5.47 Simulation studies

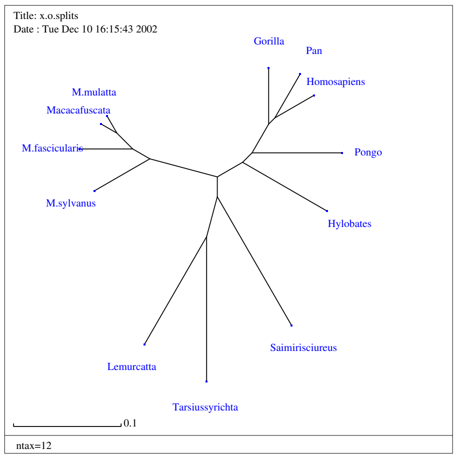
Simulation studies play an important role in bioinformatics. They are used to evaluate the performance of new algorithms on simulated datasets for which the correct answers are know.

For example, to evaluate the performance of a tree construction method, M, we can proceed as follows:

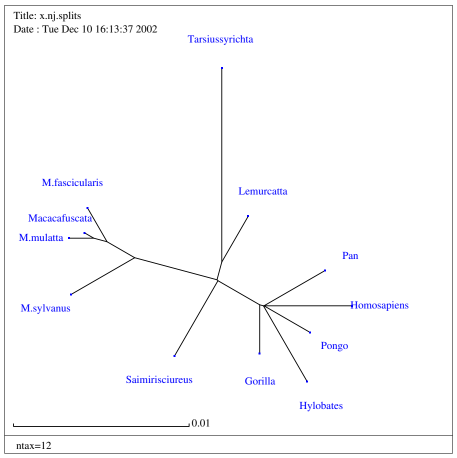
1. Select a model tree $T = (V, E, \omega)$ on X.
2. For a specified mutation rate u, generate a set of aligned sequences A on T under the Jukes-Cantor model.
3. Apply the method M to A to obtain a tree $M(A)$.
4. Compute the number of false positives and false negatives.
5. Repeat many times for many different parameters and report the average performance of the method.

Example Under the Jukes-Cantor model with $L = 1000$ and $u = 0.1$, sequences were generated on a model tree, then Hamming distances were computed, then neighbor-joining was applied:

Model tree



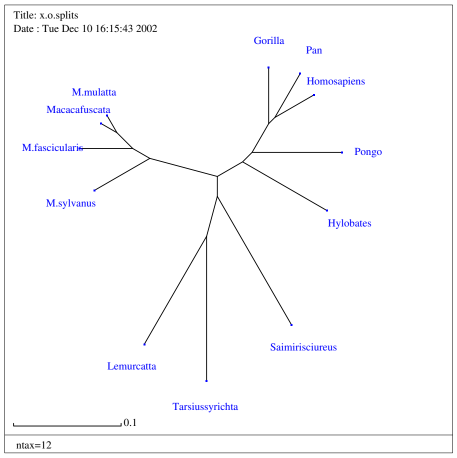
neighbor-joining tree



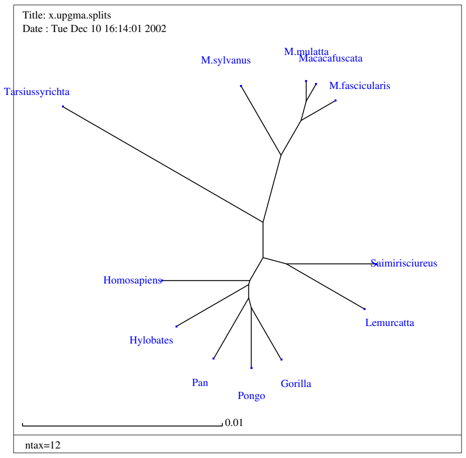
$$FP = FN = 3$$

Example Under the Jukes-Cantor model with $L = 1000$ and $u = 0.1$, sequences were generated on a model tree, then Hamming distances were computed, then UPGMA was applied:

Model tree

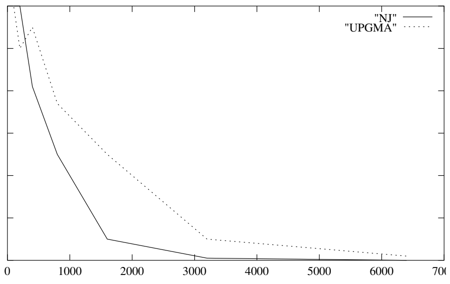


UPGMA tree

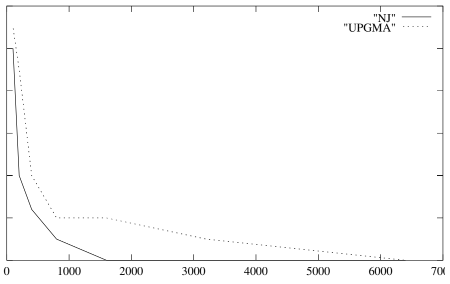


$$FP = FN = 5$$

Using the same tree and computations, here we plot the number of false positives obtaining for $u = 0.1, 0.5$ and different sequence lengths L ranging from $100 - 6400$. Each point plotted was obtained as the average of five independent simulations:



L vs FP, $u = 0.1$



L vs FP, $u = 0.4$

For this particular tree, neighbor-joining displays a slightly better performance than UPGMA. However, to obtain significant results, many trees and many different parameters must be considered.

5.48 Algorithms vs optimality criteria

In phylogenetics, the goal is to *estimate* the true evolutionary tree that generated a set of extant taxa. Tree reconstruction methods attempt to accomplish this goal by *selecting* one of the many different possible topologies.

This selection process is performed in one of two ways:

1. The tree selected by an *algorithmic* method such as neighbor-joining or UPGMA is defined by the sequence of steps that make up the algorithm. This tree does not solve any explicitly formulated optimization problem.

2. The tree selected by an *optimization* method is a solution to an explicitly formulated optimization problem such as "maximum parsimony" or "maximum likelihood". Here,

algorithms play a secondary role as a means of obtaining or approximating a solution to the optimization problem.

Distance methods are usually *algorithmically* defined and have polynomial run time. Sequence methods usually involve solving an *optimization* problem by finding an optimal tree with respect to some criterion and are usually NP-hard.

Algorithmic methods combine the computation and definition of the preferred tree into a single statement.

Optimization methods involve formulating and solving two problems:

- computing the cost for a given tree T, and
- searching through all trees to find a tree that minimizes the cost.

5.49 Maximum parsimony

The maximum parsimony method is by far the most used sequence-based tree reconstruction method.

In science, the principal of *maximum parsimony* is well known: always use the simplest, most parsimonious explanation of an observation, until new observations force one to adopt a more complex theory.

In phylogenetic analysis, the *maximum parsimony problem* is to find a phylogenetic tree that explains a given set of aligned sequences using the minimum number of substitutions.

5.50 The parsimony score of a tree

The *difference* between two sequences $x = (x_1, \ldots, x_L)$ and $y = (y_1, \ldots, y_L)$ is simply their non-normalized Hamming distance

$$\mathrm{diff}(x, y) = |\{k \mid x_k \neq y_k\}|.$$

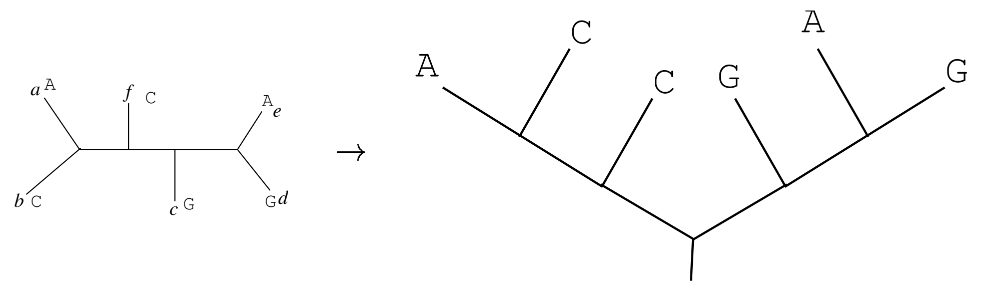
Given a multiple alignment of sequences $A = \{a_1, a_2, \ldots, a_n\}$ and a corresponding phylogenetic tree T, leaf-labeled by A.

If we assign a hypothetical ancestor sequence to every internal node in T, then we can obtain a score for T, together with this assignment, by summing over all differences $\mathrm{diff}(x, y)$, where x and y are any two sequences labeling two nodes that are joined by an edge in T.

The minimum value obtainable in this way is called the *parsimony score* $PS(T, A)$ of T and A.

Can the parsimony score of a tree T be computed efficiently?

As the parsimony score is obtained by summing over all columns, the columns are independent and so it suffices to discuss how to obtain an optimal assignment for one position:



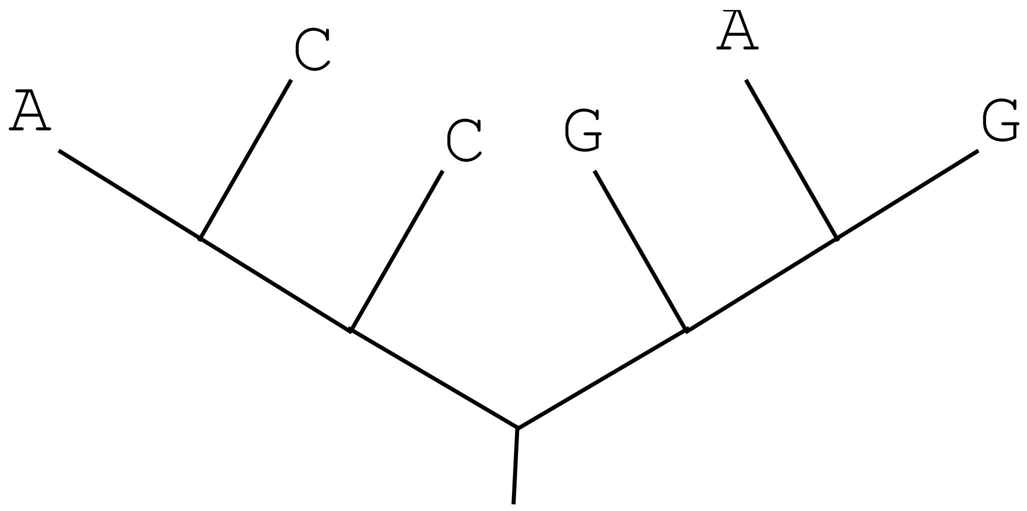
unrooted tree rooted tree

5.51 The Fitch algorithm

The following algorithm computes the parsimony score for T and a fixed column in the sequence alignment. It modifies a global score variable and is repeatedly run to obtain the total score. Initially it is called with $e = null$ and v the root node.

```
Algorithm ParsimonyScore($e, v$) (Walter Fitch 1971)
Input: A phylogenetic tree $T$ and a character $c(v)$ for each leaf $v$
Output: The parsimony score for $T$ and $c$
if $v$ is a leaf node then
  Set $R(v) = \{c(v)\}$
else
  for each edge $f_1, f_2 \neq e$ adjacent to $v$ do
       Let $w_i$ be the opposite node of $f_i$
       Call ParsimonyScore($f_i, w_i$) // to compute $R(w_i)$
  if $R(w_1) \cap R(w_2) \neq \emptyset$ then
     Set $R(v) = R(w_1) \cap R(w_2)$
  else
     Set $R(v) = R(w_1) \cup R(w_2)$ and increment the global score
end
```

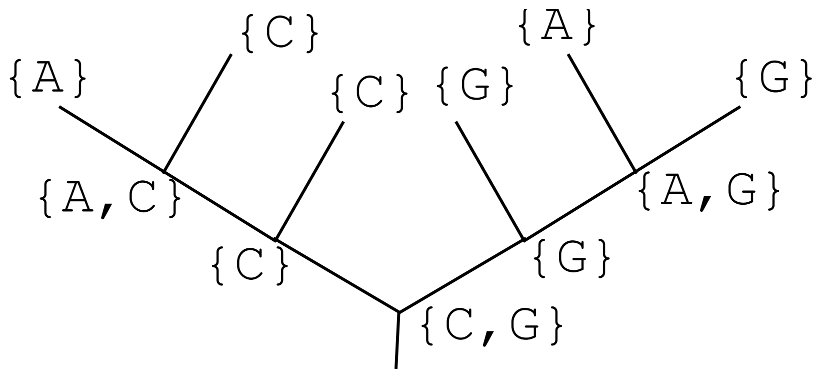
We can use the Fitch algorithm to show that the parsimony score for the following labeled tree is 3:



In total, the algorithm requires $O(nL)$ steps.

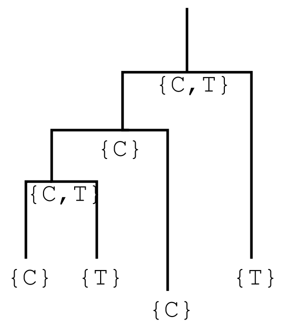
5.52 Traceback

The above algorithm computes the parsimony score. An optimal labeling of the internal nodes is obtained via traceback: starting at the root node r, we label r using any character in $R(r)$. Then, for each child w, we us the same letter, if it is contained in $R(w)$, otherwise we us any letter in $R(w)$ as label for w. We then visit the children von w etc:

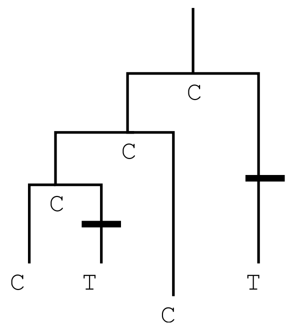


Again, the algorithm requires $O(nL)$ steps, in total.

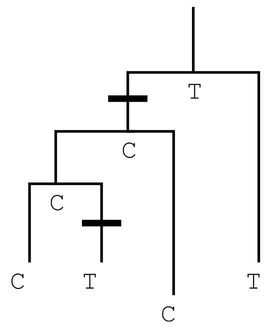
Example:



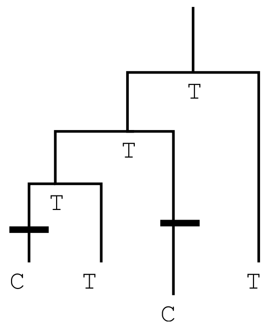
Fitch labeling



one traceback result



another traceback result



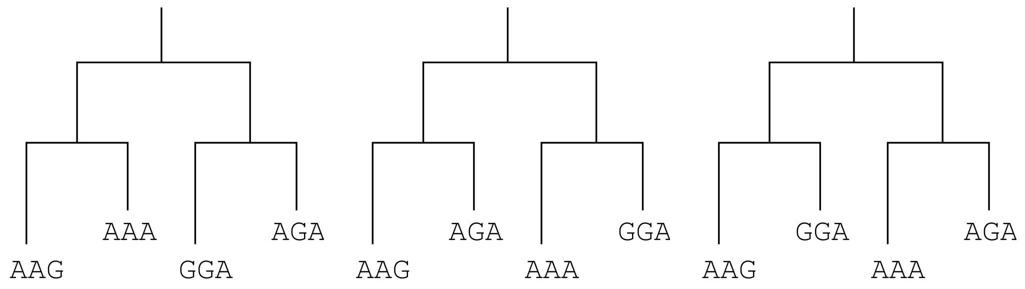
not obtainable by traceback

5.53 A simple example

Assume we are given the following four aligned sequences:

```
A A G
A A A
G G A
A G A
```

There are three possible binary topologies on four taxa:



In each tree, label all internal nodes with sequences so as to minimize the score obtained by summing all mismatches along edges. Which tree minimizes this score?

5.54 The Fitch algorithm on unrooted trees

Above, f.e.o.e. we formulated the Fitch algorithm for rooted trees. However, the minimum parsimony cost is independent of where the root is located in the tree T:

To see this, consider a root node ρ connected to two nodes v and w, and let $c(\cdot)$ denote the character assigned to a node in the Fitch algorithm:

1. If $c(v) = c(w)$, then $c(\rho) = c(v) = c(w)$, because any other choice would add 1 to the score.
2. If $c(v) \neq c(w)$, then either $c(\rho) = c(v)$ or $c(\rho) = c(w)$, and both choices contribute 1 to the score.

In both cases, we could replace ρ by a new edge e that connects v and w without changing the parsimony score of T.

Thus, we can run the Fitch algorithm on the unrooted version of T, using *any* internal node in the initial call.

5.55 The parsimony score

Given a multiple alignment $A = \{a_1, \ldots, a_n\}$, it's *parsimony score* is defined as

$$PS(A) = \min\{PS(T, A) \mid T \text{ is a phylogenetic tree on } A\}.$$

The *maximum parsimony problem* is to compute $PS(A)$.

Potentially, we need to consider all $(n-5)!!$ possible trees. Unfortunately, in general this can't be avoided and the maximum parsimony problem is known to be NP-hard.

Exhaustive enumeration of all possible tree topologies will only work for $n \leq 10$ or 11, say.

Thus, we need more efficient strategies that either solve the problem exactly, such as the branch and bound technique, or return good approximations, such as heuristic searches.

Remark: As with most biological problems, we are not only interested in the optimal solution, but would also like to know something about other near optimal solutions as well.

5.56 Branch and bound

Recall how we obtained an expression for the number $U(n)$ of unrooted phylogenetic tree topologies on n taxa:

For $n = 3$ there are three ways of adding an extra edge with a new leaf to obtain an unrooted tree on 4 leaves. This new tree has $(2n - 3) = 5$ edges and there are 5 ways to obtain a new tree with 5 leaves etc.

To be precise, one can obtain any tree T_{i+1} on $\{a_1, \dots, a_i, a_{i+1}\}$ by adding an extra edge with a leaf labeled a_{i+1} to some (unique) tree T_i on $\{a_1, \dots, a_i\}$.

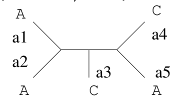
In other words, we can produce the set of *all* possible trees on n taxa by adding one leaf at a time in all possible ways, thus systematically generating a complete *enumeration tree.*

A simple, but crucial observation is that adding a new sequence a_{i+1} to a tree T_i to obtain a new tree T_{i+1} cannot lead to a smaller parsimony score.

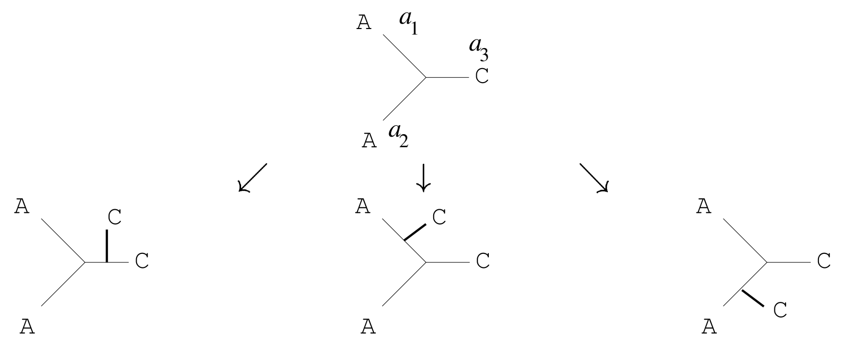
This gives rise to the following bound criterion when generating the enumeration tree: if the *local* parsimony score of the current incomplete tree T' is larger or equal to the best *global* score for any complete tree seen so far, then we do not generate or search the enumeration subtree below T'.

In practice, using branch and bound one can obtain exact solutions for data sets of twenty or more sequences, depending on the sequence length and the messiness of the data.

A good starting strategy is to first compute a tree T_0 for the data, e.g. using neighbor-joining, and then to initialize the *global* bound to the parsimony score of T_0.

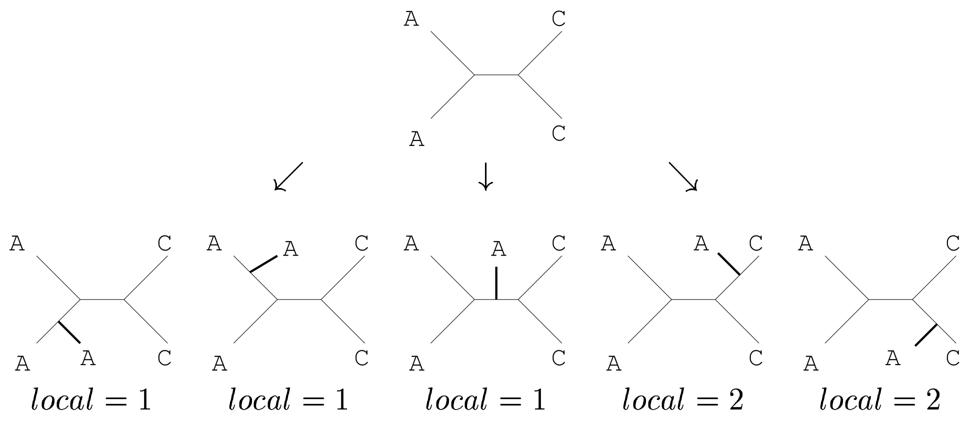
Example Assume we are given an msa $A = \{a_1, a_2, \dots, a_5\}$ and at a given position i the characters are: $a_{1i} = \texttt{A}$, $a_{2i} = \texttt{A}$, $a_{3i} = \texttt{C}$, $a_{4i} = \texttt{C}$ and $a_{5i} = \texttt{A}$. Assume that the neighbor-joining tree on A looks like this: 

and thus gives a global upper bound of *global* = 2 for position i. The first step in generating the enumeration tree is this:



local = 1 < *global* continue | *local* = 2 $\not<$ *global* don't continue | *local* = 2 $\not<$ *global* don't continue

Only the first of the three trees fulfills the bound criterion and we do not pursue the other two trees. The second step in generating the enumeration tree looks like this:



The first three trees are optimal. Note that the bound criterion in the first step reduced the number of full trees to be considered by two thirds.

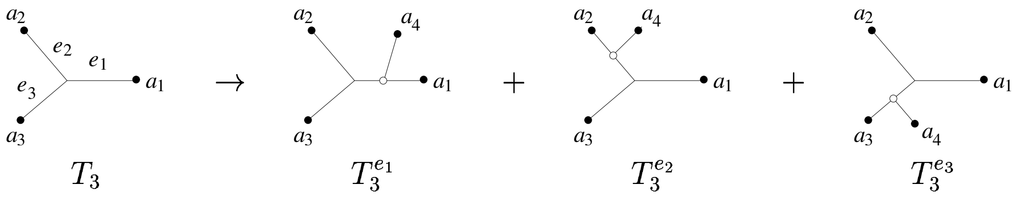
Application of branch-and-bound to evolutionary trees was first suggested by Mike Hendy and Dave Penny (1982).

5.57 The stepwise-additional heuristic

Now we discuss a simple greedy heuristic (Felsenstein 1981) for approximating the optimal tree or score. We build the tree T by adding one leaf after the other, in each step choosing the optimal position for the new leaf-edge:

Given a multiple sequence alignment $A = \{a_1, a_2, \ldots, a_n\}$. Start with a tree T_2 consisting of two leaves labeled a_1 and a_2.

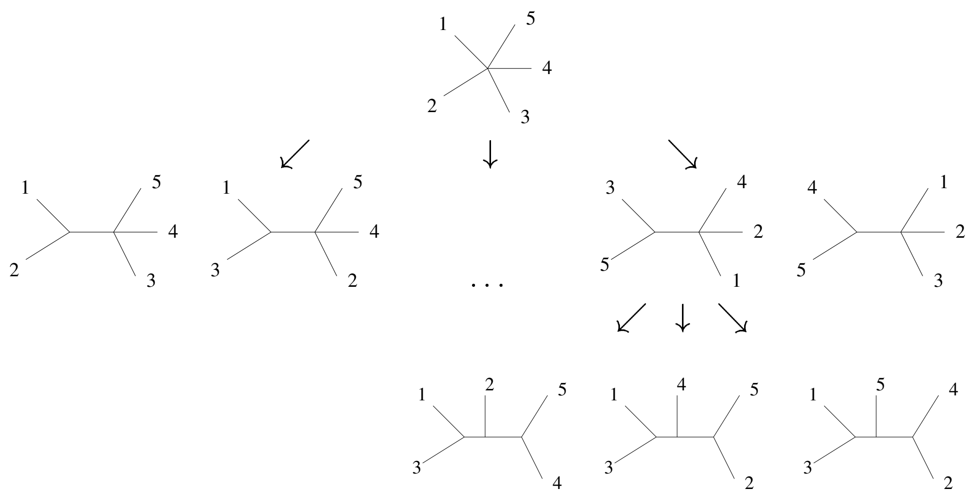
Given T_i. For each edge e in T_i, obtain a new tree T_i^e as follows: Insert a new node v in e and join it via a new edge f to a new leaf w with label a_{i+1}. Set $T_{i+1} = \arg\min\{P(T_i^e, A)\}$.



Obviously, this approach is not guaranteed to obtain an optimal result. Moreover, the result obtained will depend on the order in which sequence are processed.

5.58 The star-decomposition heuristic

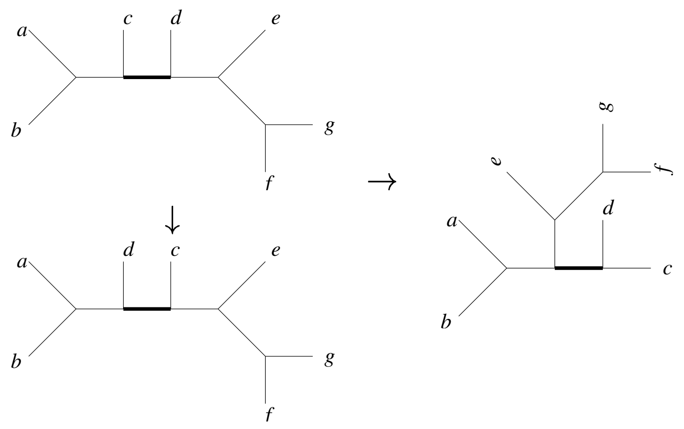
This employs a similar strategy to neighbor-joining. We start with a star tree on all n taxa. At each step, the optimality criterion is evaluated for every possible joining of a pair of lineages incident to the central node. The best tree found at one step is then used as the basis of the next step:



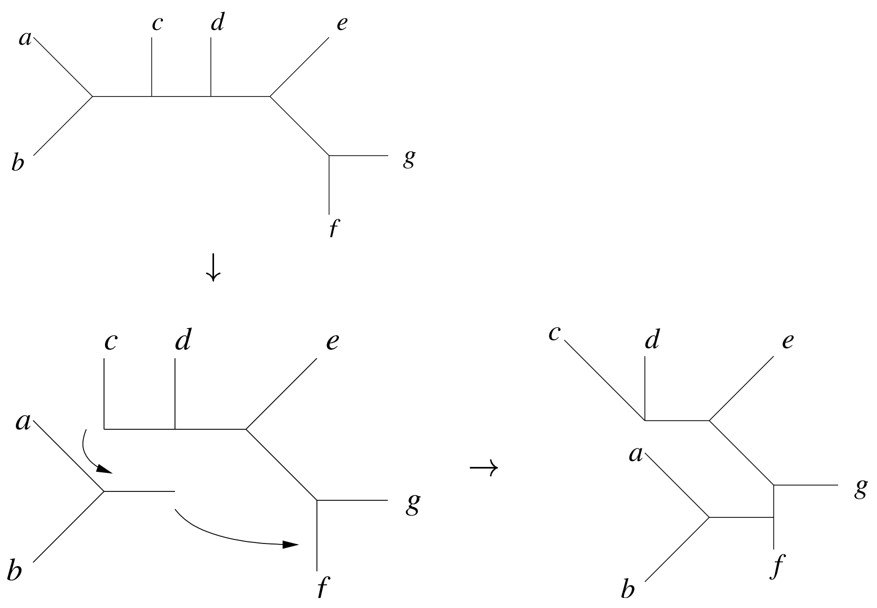
5.59 Branch swapping methods

The two heuristics just described are both very susceptible to entrapment in local optima. We now discuss a number of *branch-swapping operations* that one can use to move through the space of all trees, hopefully jumping far enough to escape from local optima.

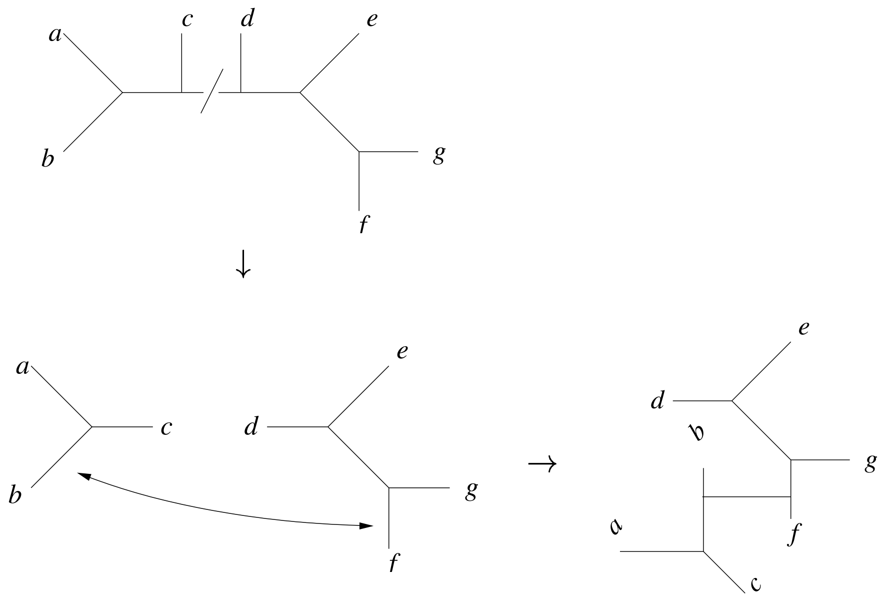
In a *nearest-neighbor interchange (NNI)*, two of the four subtrees around an edge are swapped, in two different ways:



In branch swapping by *subtree pruning and re-grafting*, a subtree is pruned from the tree and re-grafted to a different location of the tree:



In branch swapping by *tree bisection and reattachment*, the tree is bisected at an edge, yielding two subtrees. The two subtrees are then reconnected at two new positions:



5.60 Heuristic search

If the data set $A = \{a_1, \ldots, a_n\}$ is too big to be solved exactly via branch and bound, then we can use a heuristic search method in an attempt to find or approximate the optimal solution.

This involves searching through the space of all unrooted phylogenetic trees on n labels and trying to proceed toward a globally optimal one. We "move" through tree space using one or more branching-swapping techniques.

Heuristic searches employ *hill-climbing techniques*: we imagine that the "goodness" $-PS(T)$ of the solution as a landscape along which we move during the search. The general strategy is to always move upwards in the hope of reaching the top of the highest peak.

Even using the above described branch-swapping techniques, any heuristic search is in danger of "climbing the wrong mountain" and getting stuck in a local optimum. Different strategies have been developed to avoid this problem.

5.61 Simulated annealing

The *simulated annealing* method employs a *temperature* that cools over time (Van Laarhoven and Aarts, 1987). At high temperatures the search can move more easily to trees whose score is less optimal than the score of the current tree. As the temperature decreases, the search becomes more and more directed toward better trees.

I.e., let T_i denote the current tree at step i and let $z(T_i)$ denote the goodness of T_i (e.g., $-PS(T)$). In hill climbing, a move to T_{i+1} is acceptable, if $z(T_{i+1}) \geq z(T_i)$. In simulated annealing, *any* new solution is accepted with a certain probability:

$$Prob(\text{accepting solution } T_{i+1})$$

$$= \begin{cases} 1 & \text{if } z(T_{i+1}) \geq z(T_i) \\ e^{-t_i(z(T_{i+1})-z(T_i))} & \text{otherwise,} \end{cases}$$

where t_i is called the *temperature* and decreases over time.

5.62 The Great Deluge method

The *Great Deluge* method, introduced by Guenter Dueck and Tobias Scheuer (1990), employs a slowly rising water level and the search accepts any move that stays above the water level.

The probability of accepting a new solution T_{i+1} is 1, if $z(T_{i+1}) > w_i$, where w_i is a bound that increases slowly with time.

If T_{i+1} is accepted, then we update the water level by setting

$$w_{i+1} = c \times (z(T_{i+1}) - z(T_i)).$$

Typically, the constant c is usually about 0.01 to 0.05.

Another of the many heuristics is *tabu search* method (Glover, 1989) that maintains a *tabu list* of $5 - 10$ solutions recently visited and prevents the search from revisiting any solutions it has just tried.

5.63 A 2-approximation for maximum parsimony

Given a connected graph $G = (V, E)$ with edge weights $\omega(\cdot)$. A *minimum spanning tree* $ST = (V', E')$ on G is a tree with $V' = V$ and $E' \subset E$ such that $\sum_{e \in E'} \omega(e)$ is minimum.

Given a multiple alignment of sequences $A = \{a_1, \ldots, a_n\}$. Consider the complete graph $G(A)$ on n nodes whose nodes are labeled by the sequences and whose edge lengths $\omega(\cdot)$ are set to the non-normalized Hamming distances between the sequences.

Theorem Let ST be a minimum spanning (phylogenetic) tree on $G(A)$. Then ST is a 2-approximation to the most parsimonious tree on A. In other words. the parsimony score $PS(ST, A)$ is at most twice as large the optimal parsimony score $PS(A)$.

Note that a minimum spanning tree is easily computed in polynomial time. This gives us an upper bound on the parsimony length of the most parsimonious tree for a given input.

Proof Let T^* be a most parsimonious tree on A and let $2T^*$ denote the graph obtained by duplicating all edges in T^*.

By construction, each node of $2T^*$ has even degree. Hence, there exists a *Eulerian* tour C of $2T^*$ that uses every edge exactly once (as shown by Euler). From C we obtain a (shorter) tour C' of all nodes in $G(A)$ by simply visiting all nodes in the order that they appear in C.

Let $\omega(C)$ denote the sum of weights of edges in C and define $\omega(C')$ similarly. Then, clearly

$$\omega(C) = \omega(2T^*) = 2\omega(T^*),$$

since C is Eulerian, and also

$$\omega(C') \leq \omega(C),$$

since Hamming distances satisfy the triangle inequality.

Note that if we delete any single edge in C', then we obtain a path P' with

$$\omega(P') \leq \omega(C').$$

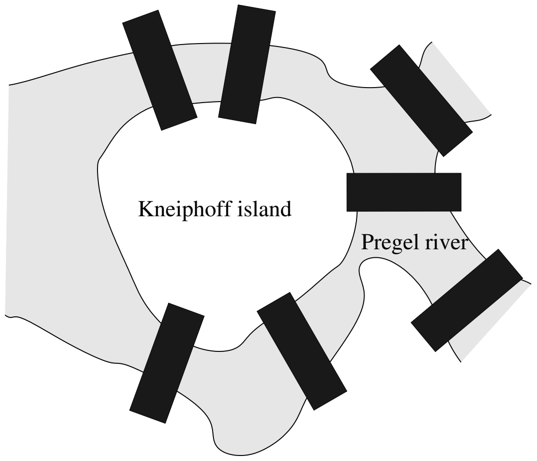
Let ST be a minimum spanning tree for $G(A)$. Since P' is also a spanning tree, we have

$$\omega(ST) \leq \omega(P') \leq \omega(C') \leq \omega(C') = 2\omega(T^*),$$

proving the claim. □

5.64 Euler Path

Leonard Euler wanted to know whether there exists a path that uses all seven bridges in Königsberg exactly once:



Birth of graph theory...

5.65 Maximum likelihood estimation (MLE)

Given a multiple alignment $A = \{a_1, \ldots, a_n\}$. Assuming a specific model of evolution M, one may attempt to *estimate* a phylogenetic tree T with edge lengths ω that *maximizes the likelihood*

$$P(A \mid T)$$

of generating the sequences $a_1, \ldots, a_n$ at the leaves of T.

A main attraction of maximum likelihood estimation (MLE) is that it provides a systematic frame-work for explicitly incorporating assumptions and knowledge about the process that generated the given data.

One potential draw-back is that any given model of evolution is only a rough estimation of real-world biological evolution. Fortunately, in practice, maximum likelihood methods have proved to be quite robust to many violations of the assumptions formulated in the models. MLE methods work very well on small data sets.

Similar to maximum parsimony, an optimal MLE tree is determined by a search in tree space. One can attempt to find an exact solution, using branch and bound techniques, or one can attempt to find a good approximate solution using a heuristic search technique...

A main draw-back of MLE is that, like maximum parsimony, finding an optimal tree is NP-hard.

In the case of maximum parsimony, we are able to compute the parsimony score for any given tree in polynomial time using the Fitch algorithm.

In the case of MLE, no such fast method of evaluating a single tree exists (unless $P = NP$), as the evaluation of a single tree is known to be NP-hard.

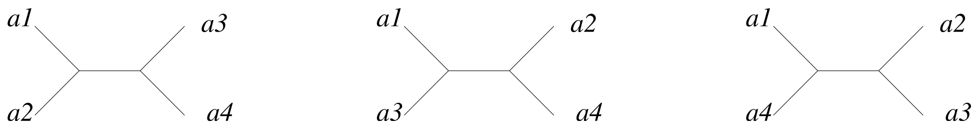
Given a multiple alignment $A = \{a_1, \ldots, a_n\}$, MLE seeks to determine the topology (evolutionary branching order) and branch lengths of the true or generating tree.

This is done under the assumption of a model of evolution, such as the Jukes-Cantor model, described above, or more general models. Such models for biological sequences are usually *time reversible* and thus the likelihood of a tree is generally independent of the location of the root.

Example Assume that we are given the following msa $A = \{a_1, \dots, a_4\}$:

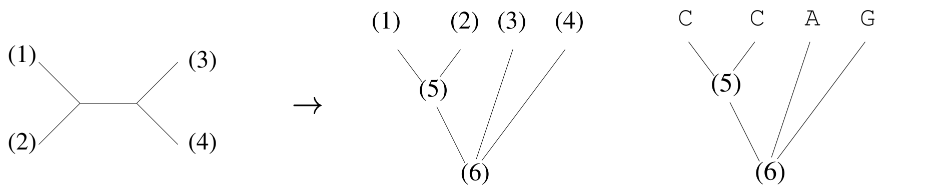
	1	2			i			N
sequence a_1	A	C	...	G	C	G	...	A
sequence a_2	A	C	...	G	C	C	...	G
sequence a_3	A	C	...	T	A	C	...	G
sequence a_4	A	C	...	T	G	G	...	A

There are three possible unrooted tree topologies on four taxa:



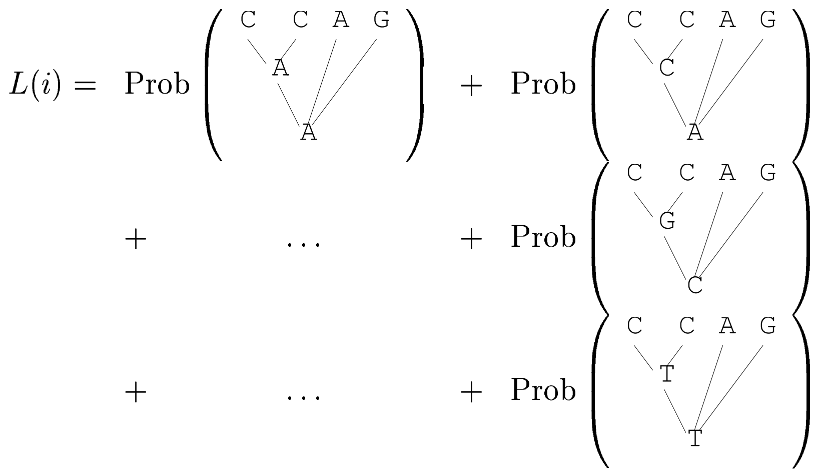
We now discuss how to compute the maximum likelihood for the first tree. The other trees are processed similarly.

For simplicity, we root the tree at an arbitrary internal node and then consider each position $i = 1, 2, \dots, N$ in the sequence:



original tree topology rooted version labeled by characters at position i

Schematically, we obtain the likelihood that *this* tree generated the characters seen at position i of the multiple alignment by summing over *all possible* labellings of the internal nodes by characters:



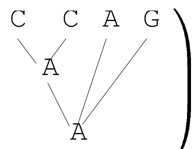
We then multiple the likelihoods obtained for each position:

$$L = L(1) \cdot L(2) \cdot \ldots \cdot L(N) = \prod_{i=1}^{N} L(i).$$

Obviously, the individual probabilities will often be very small and so we add their logarithms instead of using multiplication:

$$\ln L = \ln L(1) + \ln L(2) + \ldots + \ln L(N) = \sum_{i=1}^{N} \ln L(i).$$

Actually, the situation is more complicated as we must determine the choice of edge lengths for the given tree that produces the highest likelihood.

How do we compute e.g. Prob $\left(\right.$ 

$\left. \right)$?

Let us look at this under the Jukes-Cantor model of evolution with a fixed mutation rate u, as discussed above. For any edge e, let P and Q denote the labels at the two opposite ends of e. The probability of $P = Q$ is given by

$$\text{Prob}(Q = P \mid T) = \frac{1}{4}(1 + 3e^{-\frac{4}{3}ut}),$$

and the probability that the two characters differ is

$$1 - \text{Prob}(Q = P \mid T) = \frac{3}{4}(1 - e^{-\frac{4}{3}ut}),$$

where $t = \omega(e)$.

The total probability that *this* tree with *this* labeling of internal nodes generated the observed data at the leaves of the tree is obtained by multiplication of the probabilities for each edge.

5.66 A MLE heuristic: Quartet puzzling

Korbinian Strimmer and Arndt von Haeseler (1996) proposed a straight-forward heuristic for approximating the MLE tree that is based on the idea of computing MLE trees on quartets of taxa and then using many rounds of combination steps in which one attempts to fit a random set of quartet trees together so as to obtain a tree on the whole dataset.

Given a multiple alignment of sequences $A = \{a_1, \ldots, a_n\}$. The quartet puzzling heuristic proceeds in three steps:

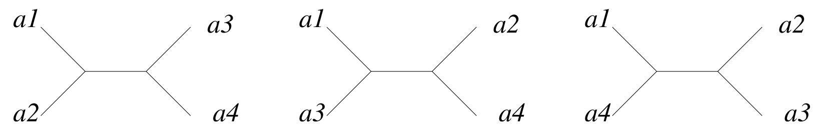
1. Compute a maximum likelihood tree on each quartet of distinct taxa $a_i, a_j, a_k, a_l \in A$.

2. Choose a random order $a_{i_1}, a_{i_2}, \ldots, a_{i_n}$ of the taxa and then use it to guide a combining or *puzzle* step that produces a tree on the whole data set from quartet trees.

3. Repeat step (2) many times to obtain trees $T_1, T_2, \ldots, T_R$. Report the *majority consensus tree* T.

5.67 Quartets

Given four taxa $\{a_1, a_2, a_3, a_4\}$, There are three possible binary topologies, each characterized by the one non-trivial split that they contain:

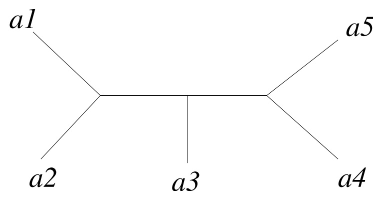


$\{\{a_1, a_2\}, \{a_3, a_4\}\}$ $\{\{a_1, a_3\}, \{a_2, a_4\}\}$ $\{\{a_1, a_4\}, \{a_2, a_3\}\}$

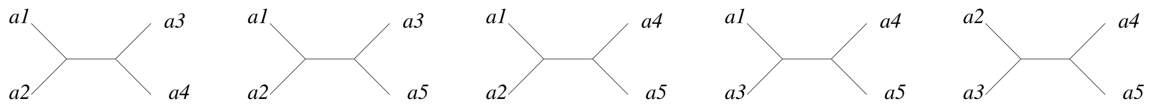
Now, let T be any phylogenetic tree that contains leaves labeled a_1, a_2, a_3, a_4, We say that T *induces* the split or tree topology $\{\{a_1, a_2\}, \{a_3, a_4\}\}$, if the exists edges in T that separates the nodes labeled a_1 and a_2 from the nodes a_3 and a_4, in which case we say that a_1 and a_2 (or a_3 and a_4) are *neighbors* (respectively).

We will sometimes abbreviate $\{\{a_1, a_2\}, \{a_3, a_4\}\}$ to $a_1, a_2 \mid a_3, a_4$.

Given this tree T:



It induces the following tree topologies on four taxa:

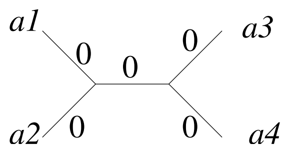


From the set of induced quartets one can reconstruct the original tree. However, in the presence of false quartet topologies as produced by any phylogenetic reconstruction method, obviously the reconstruction of the original tree can become difficult.

5.68 The initialization

In quartet puzzling, quartets are greedily combined to produce a tree for the whole data set. In an attempt to avoid the usual problems of a greedy approach, the combining step is run many times using different random orderings of the taxa.

The algorithm is initialized with the quartet tree T_4 produced on $\{a_1, a_2, a_3, a_4\}$, e.g.:



Each edge e is labeled with a *number of conflicts* $c(e)$ that is originally set to 0.

5.69 The puzzle step

In the puzzle step, the task is to decide how to insert the next taxon a_{i+1} in to the tree T_i already constructed for taxa $\{a_1, \ldots, a_i\}$. We proceed as follows:

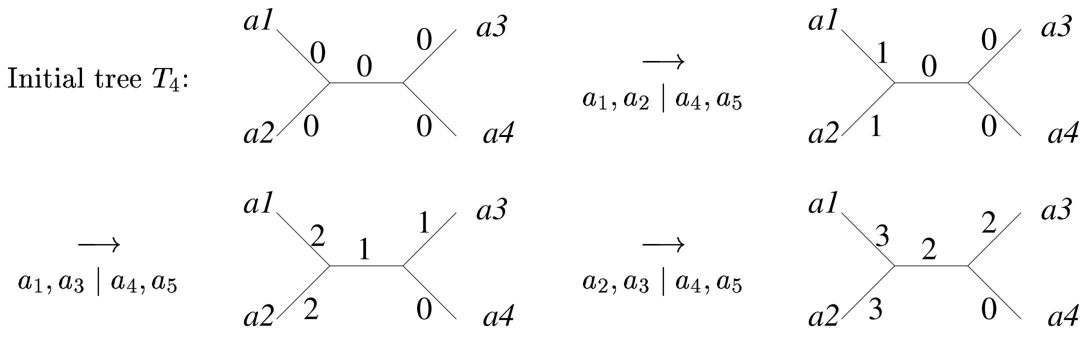
Set $c(e) = 0$ for all edges in T_i.

For each edge e in T_i we consider what would happen if we were to attach taxon a_{i+1} into the middle of e by a new edge:

We consider all quartets consisting of the taxon a_{i+1} and three taxa from $\{a_1, a_2, \ldots, a_i\}$. For each such quartet q we compare the precomputed maximum likelihood tree on q with the tree induced on q by T_i. If these two trees differ, then we increment $c(e)$ by one.

Finally, we attach e to an edge f in T_i that has a minimum number of conflicts $c(f)$, thereby obtaining T_{i+1}.

Example: We illustrate the puzzle step using the five taxon tree shown above and assume that MLE indeed produced the five listed correct quartet topologies. The puzzle step for a_5 proceeds as follows:



Hence, the puzzle step suggests that taxon a_5 should be attached in the edge adjacent to the leaf labeled a_4.

5.70 The quartet puzzling algorithm

Algorithm Quartet puzzling
Input: Aligned sequences $A = \{a_1, \ldots, a_n\}$, number of trees R
Output: Phylogenetic trees $T^1, T^2, \ldots, T^R$ on A

Initialization: Compute an MLE tree t_q for each quartet $q \subseteq A$

```
for $r = 1$ to $R$ do
    Randomly permute the order of the sequences in $A$
    Set $T_4$ equal to the precomputed MLE tree on $\{a_1, a_2, a_3, a_4\}$
    for $i = 4$ to $n$ do
        Set $c(e) = 0$ for all edges in $T_i$
        for each quartet $q \subseteq \{a_1, \ldots, a_{i+1}\}$ with $a_{i+1} \in q$ do
            Let $x, y \mid z, a_{i+1}$ be the MLE tree topology on $q$
            Let $p$ denote the path from $x$ to $y$ in $T_i$
            Increment $c(e)$ by 1 for each edge $e$ that is contained
            in $p$ or that is separated from $z$ by $p$
        Choose any edge $f$ for which $c(f)$ is minimal
        Obtain $T_{i+1}$ by attaching a new leaf with label $a_{i+1}$ to $f$
    Output $T^r := T_n$
end
```

5.71 Majority consensus

Running this algorithm produces a whole collection $T^1, \ldots, T^R$ of trees on A. To obtain a single output tree T, the quartet puzzling method computes the *majority consenus tree* for the collection of trees:

Let $\Sigma(T^i)$ denote the split encoding of T^i. The *majority-consensus splits set* for $T^1, \ldots, T^R$ is defined as the set of all splits that occur in at least half of all trees $T^1, \ldots, T^R$:

$$MC := MC(T^1, \ldots, T^R) := \left\{ S \text{ split on } A \;\middle|\; \#\{i : S \in \Sigma(T^i)\} > \frac{r}{2} \right\}.$$

Lemma The majority-consensus splits set is compatible and the corresponding tree is called the *majority consenus tree.*

Proof: Assume there exist two splits $S_1, S_2 \in MC$ that are not compatible with each other. Because each split occurs in more than half of all trees, there must exist a tree in which both occur, a contradiction. □