

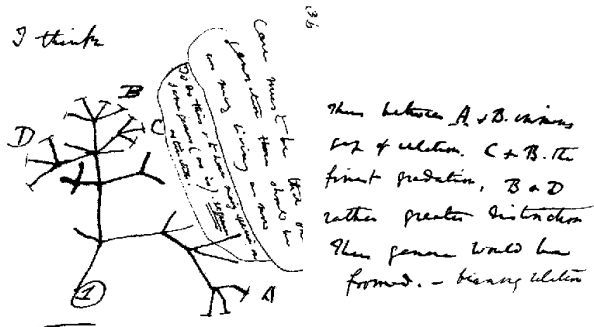
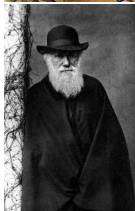
Computational Biology

Comparative Genomics and Phylogenies

Marc Hellmuth

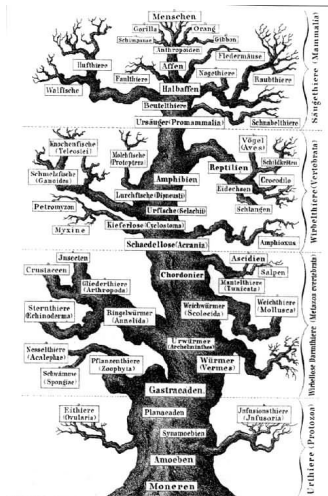
Department of Mathematics
Stockholm University

Phylogenetic Reconstruction



“I think” by Charles Darwin (1837) - One of the first evolutionary trees.

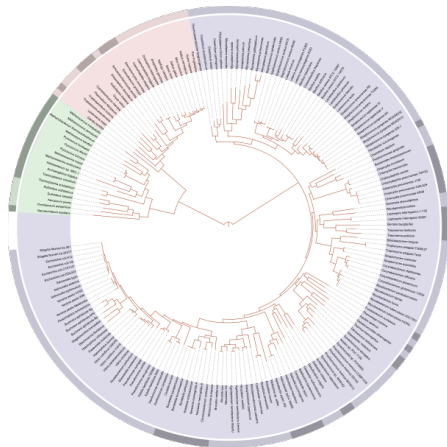
Tree of Life - A Better Picture



Ernst Haeckel, 1879

Tree of Life - A Better Picture

Relationship between species with sequenced genomes.



center = last universal ancestor of all life on earth.

three domains of life:

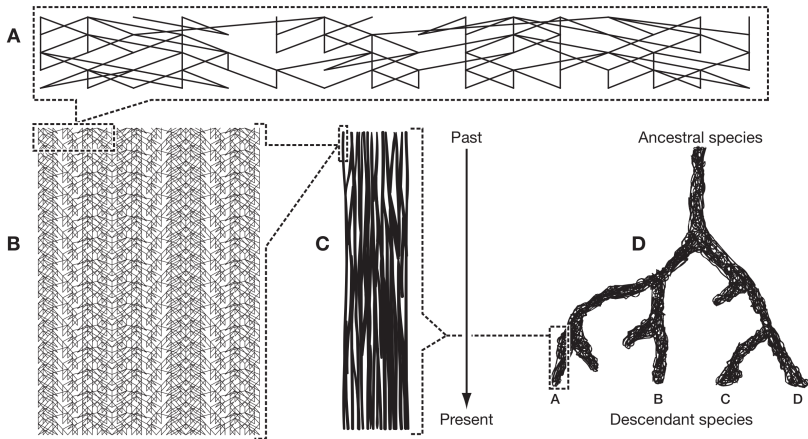
eukaryota (animals, plants and fungi);

bacteria;

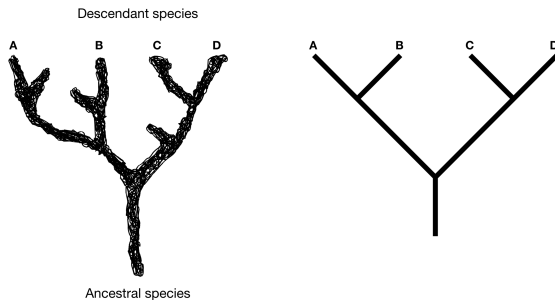
archaea.

Ciccarelli, FD (2006). "Toward automatic reconstruction of a highly resolved tree of life.". *Science*; Letunic, I (2007). "Interactive Tree Of Life (iTOL): an online tool for phylogenetic tree display and annotation.". *Bioinformatics*

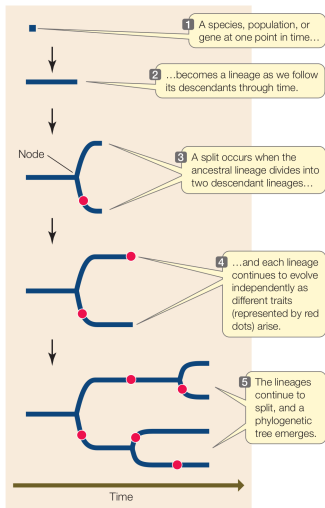
Phylogenetic Trees: The idea



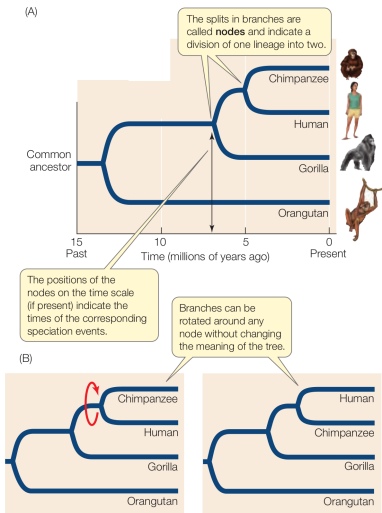
Phylogenetic Trees: The idea



extremely simplified, but powerful representation.

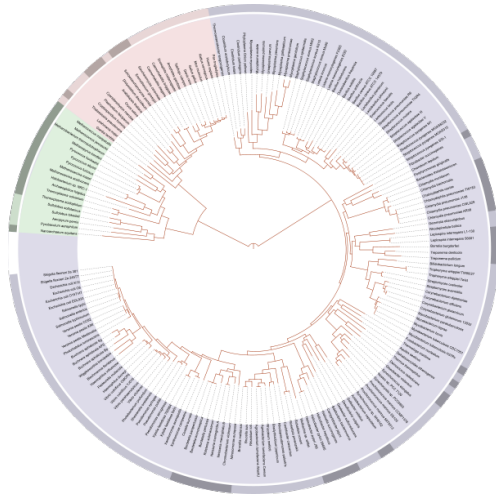


22.1 The Components of a Phylogenetic Tree Evolutionary relationships among organisms can be represented in a treelike diagram.



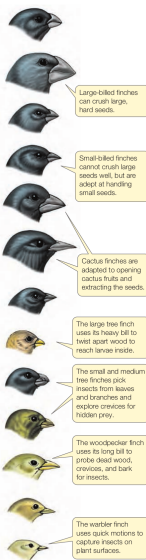
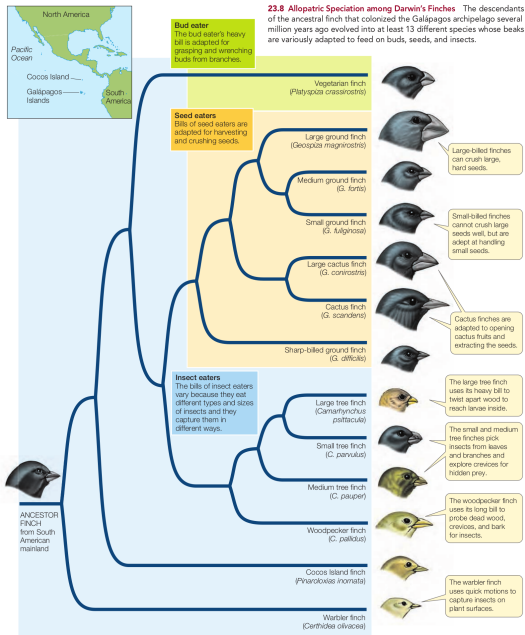
22.2 How to Read a Phylogenetic Tree (A) Phylogenetic trees can be produced with time scales, as shown here, or with no indication of time. If no time scale is shown, then the trees are only meant to depict the relative order of divergence events. (B) Lineages can be rotated around a given node, so the vertical order of taxa is largely arbitrary.

Applications: Tree of Life



Understanding how species are related to each other in evolution and finding groups of taxa

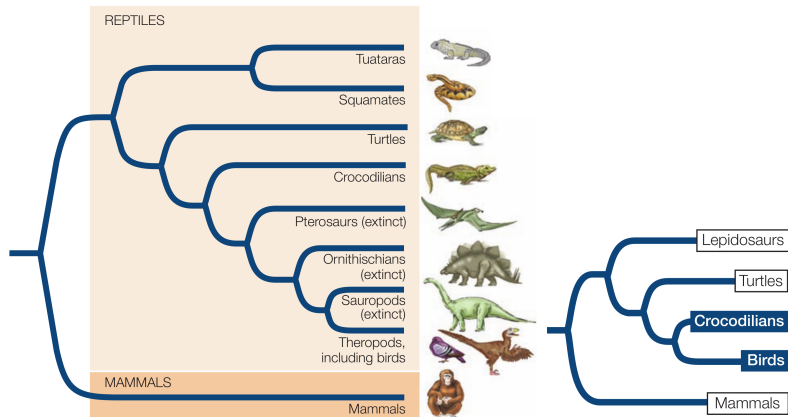
Applications: Tree of Life (a detailed view)



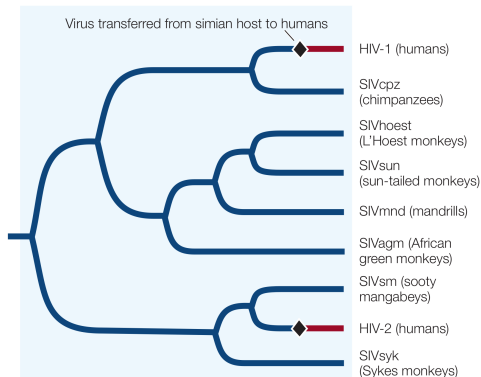
trace the history of changes and find features of interest in group or organism

Sadava et al. (2012). "LIFE: The Science of Biology (10th edition)"

Applications: Revealing surprising relationship

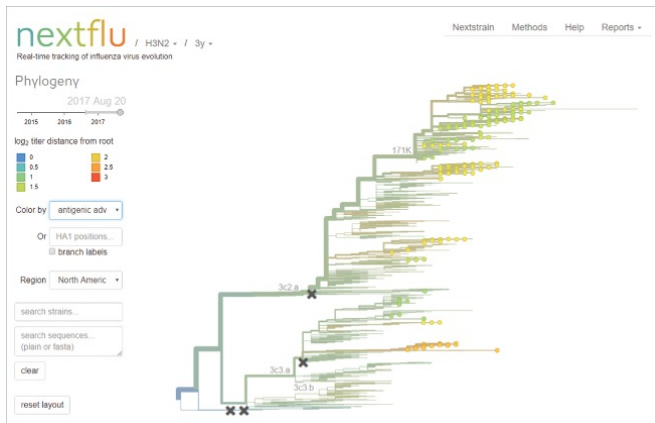


Dinosaurs did not go completely extinct and lineage survived in birds

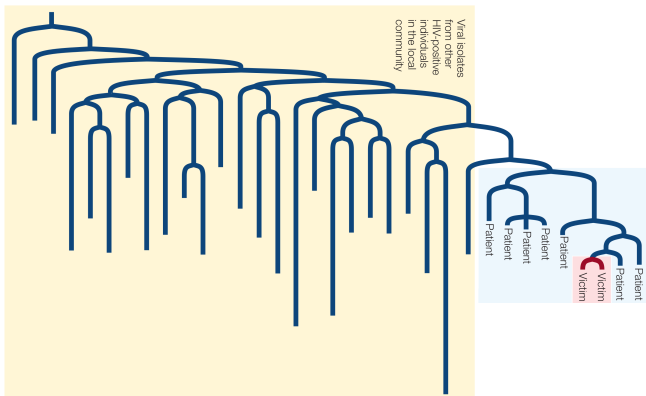


Different HIV strands
Important knowledge for finding drug treatments
(different for distinct strands)

22.8 Phylogenetic Tree of Immunodeficiency Viruses The evolutionary relationships of immunodeficiency viruses show that these viruses have been transmitted to humans from two different simian hosts: HIV-1 from chimpanzees and HIV-2 from sooty mangabeys. (SIV stands for simian immunodeficiency virus.)

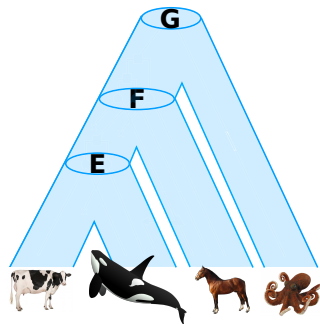


Prediction of (standart) flu vaccination

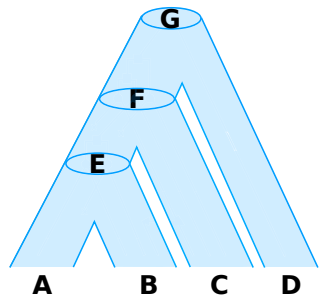


A physician who was accused of purposefully injecting blood from one of his HIV-positive patients into his former girlfriend in an attempt to kill her. The phylogenetic analysis revealed that the HIV strains present in the girlfriend were a subset of those present in the physician's patient. Based on this evidence (and other), the physician was found guilty of attempted murder by the jury.

The “true” evolutionary history

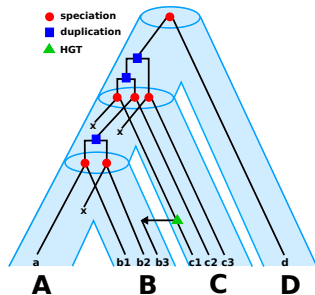


The “true” evolutionary history



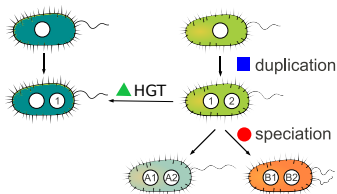
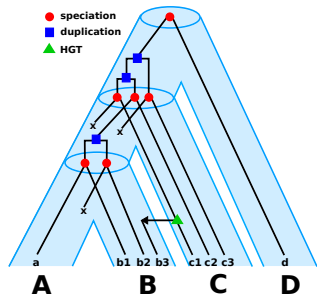
The “true” evolutionary history

- ▶ species are characterized by its genome:
a “bag of genes”
- ▶ “Genes” evolve along a *rooted* tree with unique coloring
 $t : V^0 \rightarrow M = \{\bullet, \blacksquare, \blacktriangle\}$
- ▶ “x” = gene loss



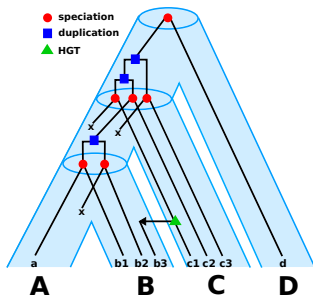
The “true” evolutionary history

- ▶ species are characterized by its genome:
a “bag of genes”
 - ▶ “Genes” evolve along a *rooted* tree with unique coloring
 $t : V^0 \rightarrow M = \{\bullet, \blacksquare, \blacktriangle\}$
 - ▶ “x” = gene loss
-
- **Gene duplication** : an offspring has two copies of a single gene of its ancestor
 - **Speciation** : two offspring species inherit the entire genome of their common ancestor
 - ▲ **HGT** : transfer of genes between organisms in a manner other than traditional reproduction and across different species



The “true” evolutionary history

- ▶ species are characterized by its genome:
a “bag of genes”
- ▶ “Genes” evolve along a *rooted* tree with unique coloring
 $t : V^0 \rightarrow M = \{\bullet, \blacksquare, \blacktriangle\}$
- ▶ “x” = gene loss



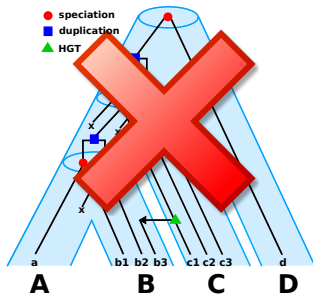
PROBLEM:

- ▶ We don't know and will never know the truth, since we cannot observe the past!

ALL proposed phylogenetic trees on real data are just approximations and reflect **hypothesis** about evolutionary histories!

The “true” evolutionary history

- ▶ species are characterized by its genome:
a “bag of genes”
- ▶ “Genes” evolve along a *rooted* tree with unique coloring
 $t : V^0 \rightarrow M = \{\bullet, \blacksquare, \blacktriangle\}$



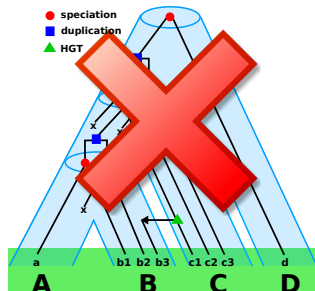
PROBLEM:

- ▶ We don't know and will never know the truth, since we cannot observe the past!

ALL proposed phylogenetic trees on real data are just approximations and reflect **hypothesis** about evolutionary histories!

The “true” evolutionary history

- ▶ species are characterized by its genome:
a “bag of genes”
- ▶ “Genes” evolve along a *rooted* tree with unique coloring
 $t : V^0 \rightarrow M = \{\bullet, \blacksquare, \blacktriangle\}$



PROBLEM:

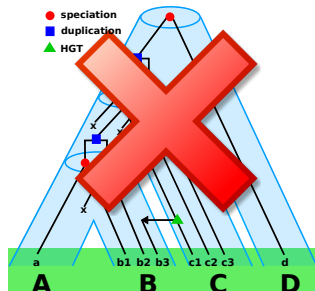
- ▶ We don't know and will never know the truth, since we cannot observe the past!

ALL proposed phylogenetic trees on real data are just approximations and reflect **hypothesis** about evolutionary histories!

- ▶ Only genetic material of extant species (“green box”) is available.

The “true” evolutionary history

- ▶ species are characterized by its genome:
a “bag of genes”
- ▶ “Genes” evolve along a *rooted* tree with unique coloring
 $t : V^0 \rightarrow M = \{\bullet, \blacksquare, \blacktriangle\}$



PROBLEM:

- ▶ We don't know and will never know the truth, since we cannot observe the past!

ALL proposed phylogenetic trees on real data are just approximations and reflect **hypothesis** about evolutionary histories!

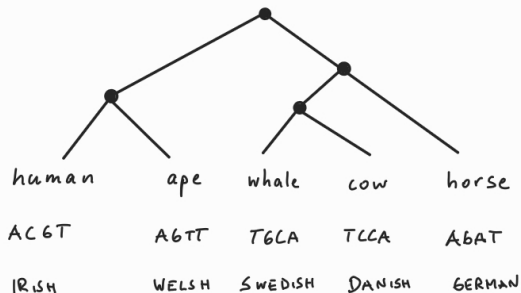
- ▶ Only genetic material of extant species ("green box") is available.

What now?

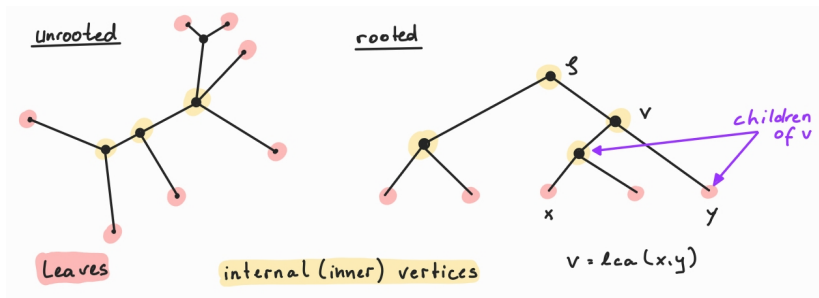
Aim: Assemble a tree representing a hypothesis about the evolutionary history of a set of genes, species or other taxa.

Trees are "good" approximation (does not work if one has hybridization)

TAXA:



Some Terminology



- ▶ Tree = connected, acyclic graph
- ▶ **phylogenetic** (We are interested in branching events!)
 - ▶ unrooted: degree $\geq 3 \forall$ inner vertices
 - ▶ rooted: at least 2 children \forall inner vertices
- ▶ Fully-resolved or **binary**
 - ▶ unrooted: degree = 3 \forall inner vertices
 - ▶ rooted: exactly 2 children \forall inner vertices
- ▶ \preceq_T in rooted trees and $\text{lca}_T(x, y)$

Depending on the application, phylogenetic trees may:

- ▶ be rooted or unrooted
- ▶ have weighted or unweighted edges / vertices
- ▶ labeled vertices / edges
- ▶ have bounded degree
(maximum nr of children of each internal node)
- ▶ ...

The problem in practice

- ▶ Inference of the gene or species tree T is a classical problem of molecular phylogenetics.

In practice it can only be solved approximately.

- ▶ Only leaves of tree corresponding to extant (currently “observable”) taxa is available.
- ▶ **Reconstructed trees do only provide a hypothesis about history!**

Lemma

There are $(2n - 3)!!$ rooted trees $(2n - 5)!!$ unrooted trees with n leaves labeled from $1, \dots, n$.

$$(m)!! := \prod_{k=0}^{\lceil \frac{m}{2} \rceil - 1} (m - 2k) = m(m - 2)(m - 4) \cdots$$

	n	3	4	5	6	10	20
Exmpl:	unrooted	1	3	15	105	2'027'025	$2.22 \cdot 10^{20}$
	rooted	3	15	105	945	34'459'425	$8.20 \cdot 10^{21}$

Enumeration / exhaustive search is no option!

The problem in practice

- ▶ Inference of the gene or species tree T is a classical problem of molecular phylogenetics.

In practice it can only be solved approximately.

- ▶ Only leaves of tree corresponding to extant (currently “observable”) taxa is available.
- ▶ **Reconstructed trees do only provide a hypothesis about history!**

Lemma

There are $(2n - 3)!!$ rooted trees $(2n - 5)!!$ unrooted trees with n leaves labeled from $1, \dots, n$.

$$(m)!! := \prod_{k=0}^{\lceil \frac{m}{2} \rceil - 1} (m - 2k) = m(m - 2)(m - 4) \cdots .$$

n		3	4	5	6	10	20
Exmpl:	unrooted	1	3	15	105	2'027'025	$2.22 \cdot 10^{20}$
	rooted	3	15	105	945	34'459'425	$8.20 \cdot 10^{21}$

Enumeration / exhaustive search is no option!

Aim: Assemble a tree representing a hypothesis about the evolutionary history of a set of genes, species or other taxa.

Methods:

- ▶ **Distance Based** e.g.:
 - ▶ Ultrametric Tree Reconstruction (UPGMA)
 - ▶ Additive Tree Reconstruction (Neighbor-Joining)
- ▶ **Character Based** e.g.:
 - ▶ Parsimony Methods (Fitch- and Sankoff Algorithm)
 - ▶ Maximum Likelihood (not part here)
- ▶ **Consensus Methods** e.g.:
 - ▶ Supertree from subtrees (BUILD)

Distance-Based Methods

Unweighted **P**air **G**roup **M**ethod with **A**rithmetic **M**ean

is a bottom-up hierarchical clustering method developed by Sokal and Michener (1958)

Given is a Distance matrix $D: X \times X \rightarrow \mathbb{R}$ on a set $X = \{x_1, \dots, x_n\}$ of taxa.

Init clusters $C_i = \{x_i\}$, $1 \leq i \leq n$

In each step of UPGMA merge the two *closest* clusters C_i, C_j into new cluster $C_{new} = C_i \cup C_j$

After merging re-compute distances for all clusters $C \neq C_{new}$:

$$D(C_{new}, C) = \frac{1}{|C_{new}||C|} \sum_{x \in C_{new}, y \in C} D(x, y)$$

= the mean distance between the taxa in C_{new} and C .

Repeat until one cluster remains

Unweighted Pair Group Method with Arithmetic Mean

is a bottom-up hierarchical clustering method developed by Sokal and Michener (1958)

Given is a Distance matrix $D: X \times X \rightarrow \mathbb{R}$ on a set $X = \{x_1, \dots, x_n\}$ of taxa.

Init clusters $C_i = \{x_i\}$, $1 \leq i \leq n$

In each step of UPGMA merge the two *closest* clusters C_i, C_j into new cluster $C_{new} = C_i \cup C_j$

After merging re-compute distances for all clusters $C \neq C_{new}$:

$$D(C_{new}, C) = \frac{1}{|C_{new}| |C|} \sum_{x \in C_{new}, y \in C} D(x, y)$$

= the mean distance between the taxa in C_{new} and C .

Repeat until one cluster remains

Unweighted Pair Group Method with Arithmetic Mean

is a bottom-up hierarchical clustering method developed by Sokal and Michener (1958)

Given is a Distance matrix $D: X \times X \rightarrow \mathbb{R}$ on a set $X = \{x_1, \dots, x_n\}$ of taxa.

Init clusters $C_i = \{x_i\}$, $1 \leq i \leq n$

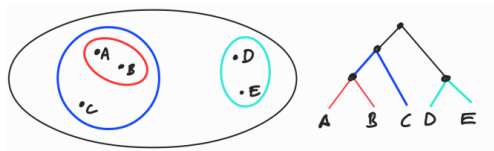
In each step of UPGMA merge the two *closest* clusters C_i, C_j into new cluster $C_{new} = C_i \cup C_j$

After merging re-compute distances for all clusters $C \neq C_{new}$:

$$D(C_{new}, C) = \frac{1}{|C_{new}||C|} \sum_{x \in C_{new}, y \in C} D(x, y)$$

= the mean distance between the taxa in C_{new} and C .

Repeat until one cluster remains



Example Whiteboard

Unweighted Pair Group Method with Arithmetic Mean

is a bottom-up hierarchical clustering method developed by Sokal and Michener (1958)

Given is a Distance matrix $D: X \times X \rightarrow \mathbb{R}$ on a set $X = \{x_1, \dots, x_n\}$ of taxa.

Init clusters $C_i = \{x_i\}$, $1 \leq i \leq n$

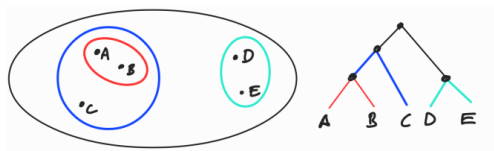
In each step of UPGMA merge the two *closest* clusters C_i, C_j into new cluster $C_{new} = C_i \cup C_j$

After merging re-compute distances for all clusters $C \neq C_{new}$:

$$D(C_{new}, C) = \frac{1}{|C_{new}| |C|} \sum_{x \in C_{new}, y \in C} D(x, y)$$

= the mean distance between the taxa in C_{new} and C .

Repeat until one cluster remains



Example Whiteboard

Unweighted Pair Group Method with Arithmetic Mean

is a bottom-up hierarchical clustering method developed by Sokal and Michener (1958)

Given is a Distance matrix $D: X \times X \rightarrow \mathbb{R}$ on a set $X = \{x_1, \dots, x_n\}$ of taxa.

Init clusters $C_i = \{x_i\}$, $1 \leq i \leq n$

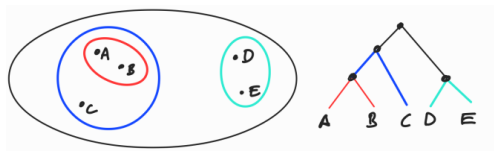
In each step of UPGMA merge the two *closest* clusters C_i, C_j into new cluster $C_{new} = C_i \cup C_j$

After merging re-compute distances for all clusters $C \neq C_{new}$:

$$D(C_{new}, C) = \frac{1}{|C_{new}| |C|} \sum_{x \in C_{new}, y \in C} D(x, y)$$

= the mean distance between the taxa in C_{new} and C .

Repeat until one cluster remains



Unweighted Pair Group Method with Arithmetic Mean

is a bottom-up hierarchical clustering method developed by Sokal and Michener (1958)

Given is a Distance matrix $D: X \times X \rightarrow \mathbb{R}$ on a set $X = \{x_1, \dots, x_n\}$ of taxa.

Init clusters $C_i = \{x_i\}$, $1 \leq i \leq n$

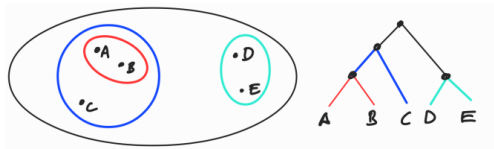
In each step of UPGMA merge the two *closest* clusters C_i, C_j into new cluster $C_{new} = C_i \cup C_j$

After merging re-compute distances for all clusters $C \neq C_{new}$:

$$D(C_{new}, C) = \frac{1}{|C_{new}| |C|} \sum_{x \in C_{new}, y \in C} D(x, y)$$

= the mean distance between the taxa in C_{new} and C .

Repeat until one cluster remains



Unweighted Pair Group Method with Arithmetic Mean

is a bottom-up hierarchical clustering method developed by Sokal and Michener (1958)

Given is a Distance matrix $D: X \times X \rightarrow \mathbb{R}$ on a set $X = \{x_1, \dots, x_n\}$ of taxa.

Init clusters $C_i = \{x_i\}$, $1 \leq i \leq n$

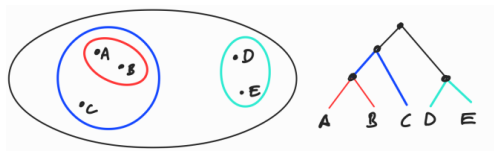
In each step of UPGMA merge the two *closest* clusters C_i, C_j into new cluster $C_{new} = C_i \cup C_j$

After merging re-compute distances for all clusters $C \neq C_{new}$:

$$D(C_{new}, C) = \frac{1}{|C_{new}| |C|} \sum_{x \in C_{new}, y \in C} D(x, y)$$

= the mean distance between the taxa in C_{new} and C .

Repeat until one cluster remains



Example Whiteboard

Two sets A, B do **not overlap** if $A \cap B \in \{A, B, \emptyset\}$.

A set \mathbb{C} of sets is a hierarchy if no two elements in \mathbb{C} overlap.

For a rooted tree T on X put

$$L(v) := \{x \in X \mid x \preceq_T v\}, v \in V(T)$$

and

$$\mathcal{C}(T) = \{L(v) \mid v \in V(T)\}$$

Exercise: $\mathcal{C}(T)$ and \mathbb{C} as computed with UPGMA are hierarchies.

Example Whiteboard

Theorem

Let \mathbb{C} be a collection of non-empty subsets of X . Then, there is a rooted phylogenetic tree T with $\mathcal{C}(T) = \mathbb{C}$ if and only if \mathbb{C} is a hierarchy. Up to isomorphism, this tree T is unique.

without proof

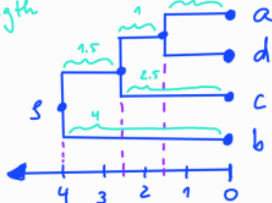
Example Whiteboard

Keep track of branch-length:

Each step of merging to clusters C, C' means that we create a new vertex v in the underlying tree such that $L(v) = C \cup C'$ and the distance from v to any leaf $x \in L(v)$ is supposed to be the same:

$$\delta(v) = D(C, C')/2$$

Branch-length



Perfectly
represented
by
tree

	a	b	c	d
a	0	8	5	3
b		0	8	8
c			0	5
d				0

Example Whiteboard

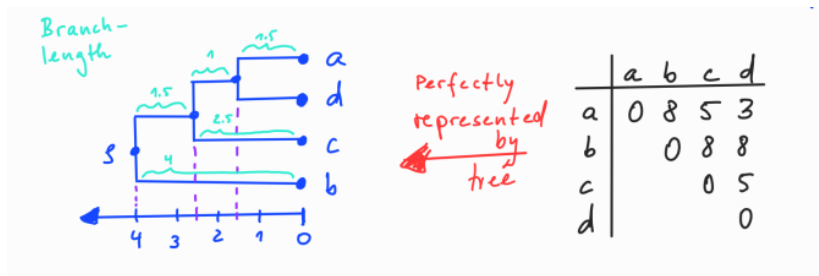
Q: Does this always work in such a perfect way?

A: No, it depends on the distances D !

Keep track of branch-length:

Each step of merging to clusters C, C' means that we create a new vertex v in the underlying tree such that $L(v) = C \cup C'$ and the distance from v to any leaf $x \in L(v)$ is supposed to be the same:

$$\delta(v) = D(C, C')/2$$



Example Whiteboard

Q: Does this always work in such a perfect way?

A: No, it depends on the distances D !

A tree T with branch length δ is an **ultrametric tree** if all leaves have the same distance to the root and $u \prec_T v$ implies $\delta(u) \leq \delta(v)$.

(T, δ) represents map $D: X \times X \rightarrow \mathbb{R}_{\geq 0}$ iff $\delta(v) = \frac{1}{2}D(x, y)$ for all $x, y \in X$ with $v = \text{lca}_T(x, y)$.

$D: X \times X \rightarrow \mathbb{R}_{\geq 0}$ is an **ultrametric** if for all $x, y, z \in X$ it holds that

- (1) $D(x, y) = 0 \iff x = y$
- (2) $D(x, y) = D(y, x)$
- (3) $D(x, y) \leq \max\{D(x, z), D(y, z)\}$

Lemma (3-point condition)

A symmetric map $D: X \times X \rightarrow \mathbb{R}_{\geq 0}$ is an ultrametric if and only if the two largest distances among $D(x, y), D(x, z), D(y, z)$ are equal

proof - whiteboard

Theorem

There is an ultrametric tree (T, δ) that represents D if and only if D is an ultrametric.

proof - whiteboard

A tree T with branch length δ is an **ultrametric tree** if all leaves have the same distance to the root and $u \prec_T v$ implies $\delta(u) \leq \delta(v)$.

(T, δ) **represents** map $D: X \times X \rightarrow \mathbb{R}_{\geq 0}$ iff $\delta(v) = \frac{1}{2}D(x, y)$ for all $x, y \in X$ with $v = \text{lca}_T(x, y)$.

$D: X \times X \rightarrow \mathbb{R}_{\geq 0}$ is an **ultrametric** if for all $x, y, z \in X$ it holds that

- (1) $D(x, y) = 0 \iff x = y$
- (2) $D(x, y) = D(y, x)$
- (3) $D(x, y) \leq \max\{D(x, z), D(y, z)\}$

Lemma (3-point condition)

A symmetric map $D: X \times X \rightarrow \mathbb{R}_{\geq 0}$ is an ultrametric if and only if the two largest distances among $D(x, y), D(x, z), D(y, z)$ are equal

proof - whiteboard

Theorem

There is an ultrametric tree (T, δ) that represents D if and only if D is an ultrametric.

proof - whiteboard

A tree T with branch length δ is an **ultrametric tree** if all leaves have the same distance to the root and $u \prec_T v$ implies $\delta(u) \leq \delta(v)$.

(T, δ) **represents** map $D: X \times X \rightarrow \mathbb{R}_{\geq 0}$ iff $\delta(v) = \frac{1}{2}D(x, y)$ for all $x, y \in X$ with $v = \text{lca}_T(x, y)$.

$D: X \times X \rightarrow \mathbb{R}_{\geq 0}$ is an **ultrametric** if for all $x, y, z \in X$ it holds that

- (1) $D(x, y) = 0 \iff x = y$
- (2) $D(x, y) = D(y, x)$
- (3) $D(x, y) \leq \max\{D(x, z), D(y, z)\}$

Lemma (3-point condition)

A symmetric map $D: X \times X \rightarrow \mathbb{R}_{\geq 0}$ is an ultrametric if and only if the two largest distances among $D(x, y), D(x, z), D(y, z)$ are equal

proof - whiteboard

Theorem

There is an ultrametric tree (T, δ) that represents D if and only if D is an ultrametric.

proof - whiteboard

A tree T with branch length δ is an **ultrametric tree** if all leaves have the same distance to the root and $u \prec_T v$ implies $\delta(u) \leq \delta(v)$.

(T, δ) **represents** map $D: X \times X \rightarrow \mathbb{R}_{\geq 0}$ iff $\delta(v) = \frac{1}{2}D(x, y)$ for all $x, y \in X$ with $v = \text{lca}_T(x, y)$.

$D: X \times X \rightarrow \mathbb{R}_{\geq 0}$ is an **ultrametric** if for all $x, y, z \in X$ it holds that

(1) $D(x, y) = 0 \iff x = y$

(2) $D(x, y) = D(y, x)$

(3) $D(x, y) \leq \max\{D(x, z), D(y, z)\}$

Lemma (3-point condition)

A symmetric map $D: X \times X \rightarrow \mathbb{R}_{\geq 0}$ is an ultrametric if and only if the two largest distances among $D(x, y), D(x, z), D(y, z)$ are equal

proof - whiteboard

Theorem

There is an ultrametric tree (T, δ) that represents D if and only if D is an ultrametric.

proof - whiteboard

A tree T with branch length δ is an **ultrametric tree** if all leaves have the same distance to the root and $u \prec_T v$ implies $\delta(u) \leq \delta(v)$.

(T, δ) **represents** map $D: X \times X \rightarrow \mathbb{R}_{\geq 0}$ iff $\delta(v) = \frac{1}{2}D(x, y)$ for all $x, y \in X$ with $v = \text{lca}_T(x, y)$.

$D: X \times X \rightarrow \mathbb{R}_{\geq 0}$ is an **ultrametric** if for all $x, y, z \in X$ it holds that

(1) $D(x, y) = 0 \iff x = y$

(2) $D(x, y) = D(y, x)$

(3) $D(x, y) \leq \max\{D(x, z), D(y, z)\}$

Lemma (3-point condition)

A symmetric map $D: X \times X \rightarrow \mathbb{R}_{\geq 0}$ is an ultrametric if and only if the two largest distances among $D(x, y), D(x, z), D(y, z)$ are equal

proof - whiteboard

Theorem

There is an ultrametric tree (T, δ) that represents D if and only if D is an ultrametric.

proof - whiteboard

Drawbacks:

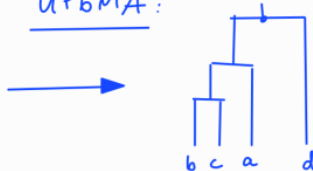
Constant Molecular-Clock Assumption: The “speed of evolution”, i.e., mutation rates are constant along all branches.



branch length $\hat{=}$ rates

D	a	b	c	d
a	0	7	5	6
b		0	4	9
c			0	7
d				0

UPGMA:



Neighbor-Joining (NJ)

- ▶ NJ is another very sophisticated distance-based method to compute unrooted trees developed by Saitou and Nei (1987)
- ▶ NJ does not make a Constant Molecular-Clock Assumption
- ▶ NJ is based on the concept of minimum-evolution, i.e., the resulting tree will have minimum total branch length.
- ▶ The idea is simple but the details are by far not trivial!

Main Idea:

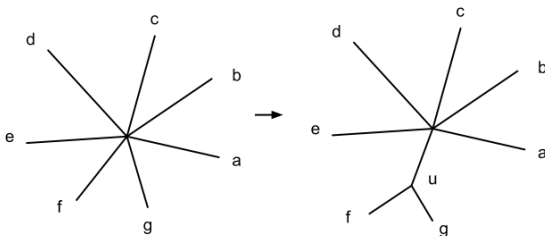
Start with star-tree and stepwise separate verices that are quite close to each other and at the same time together quite far away from the remaining leaves until a fully-resolved unrooted tree has been built.

Neighbor-Joining (NJ)

- ▶ NJ is another very sophisticated distance-based method to compute unrooted trees developed by Saitou and Nei (1987)
- ▶ NJ does not make a Constant Molecular-Clock Assumption
- ▶ NJ is based on the concept of minimum-evolution, i.e., the resulting tree will have minimum total branch length.
- ▶ The idea is simple but the details are by far not trivial!

Main Idea:

Start with star-tree and stepwise separate vertices that are **quite close to each other** and at the same time **together quite far away** from the remaining leaves until a fully-resolved unrooted tree has been built.



Neighbor-Joining (NJ)

For a given distance matrix $D: X \times X \rightarrow \mathbb{R}$ with $n = |X|$, the matrix D^* denotes the **NJ-matrix** that is defined by:

$$D_{i,j}^* = (n - 2)D_{i,j} - \text{TotalDist}_D(i) - \text{TotalDist}_D(j)$$

where $\text{TotalDist}_D(x) = \sum_{y \in X \setminus \{x\}} D(x, y)$ for all $x \in X$.

Intuition:

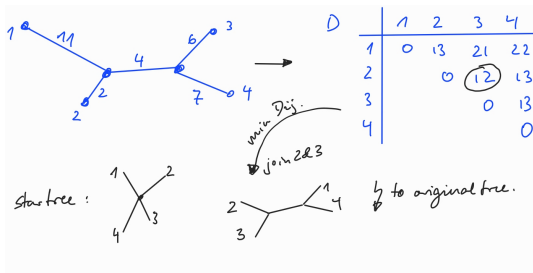
Neighbor-Joining (NJ)

For a given distance matrix $D: X \times X \rightarrow \mathbb{R}$ with $n = |X|$, the matrix D^* denotes the **NJ-matrix** that is defined by:

$$D_{i,j}^* = (n - 2)D_{i,j} - \text{TotalDist}_D(i) - \text{TotalDist}_D(j)$$

where $\text{TotalDist}_D(x) = \sum_{y \in X \setminus \{x\}} D(x, y)$ for all $x \in X$.

Intuition:



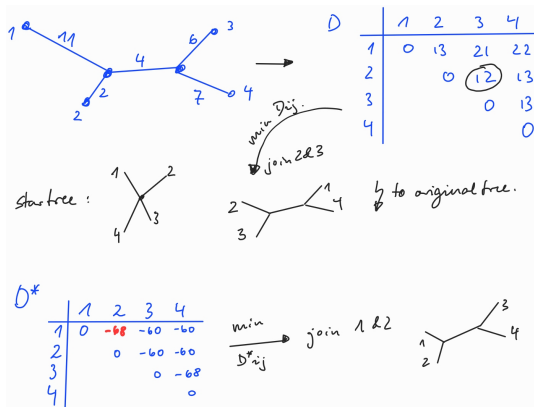
Neighbor-Joining (NJ)

For a given distance matrix $D: X \times X \rightarrow \mathbb{R}$ with $n = |X|$, the matrix D^* denotes the **NJ-matrix** that is defined by:

$$D_{i,j}^* = (n-2)D_{i,j} - \text{TotalDist}_D(i) - \text{TotalDist}_D(j)$$

where $\text{TotalDist}_D(x) = \sum_{y \in X \setminus \{x\}} D(x,y)$ for all $x \in X$.

Intuition:



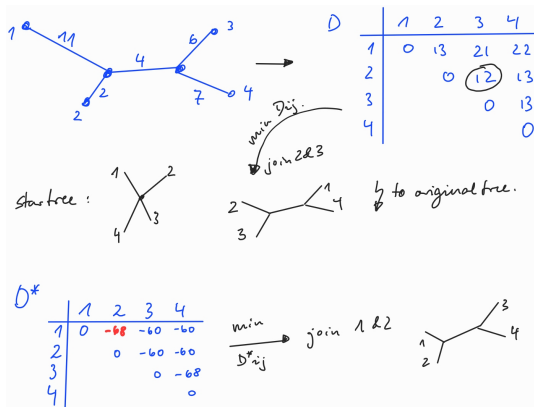
Neighbor-Joining (NJ)

For a given distance matrix $D: X \times X \rightarrow \mathbb{R}$ with $n = |X|$, the matrix D^* denotes the **NJ-matrix** that is defined by:

$$D_{i,j}^* = (n-2)D_{i,j} - \text{TotalDist}_D(i) - \text{TotalDist}_D(j)$$

where $\text{TotalDist}_D(x) = \sum_{y \in X \setminus \{x\}} D(x,y)$ for all $x \in X$.

Intuition:



D^* is "common net divergence"

Keep track of branch-length:

Want to have for all edges incident to the newly-merged leaves i and j the corresponding branch-length δ_i and δ_j

To this end define: $\Delta_{i,j} = \frac{\text{TotalDist}_D(i) - \text{TotalDist}_D(j)}{n-2}$

Small computation (whiteboard) shows that

$$\delta_i = \frac{1}{2}(D_{i,j} + \Delta_{i,j}) \text{ and } \delta_j = \frac{1}{2}(D_{i,j} - \Delta_{i,j})$$

Hence, taking total length in a smart way allows us to compute the single branch length in each step.

Algorithm:

Neighbor-Joining(D)

If (D is 1×1 matrix) then stop

Construct D^* from D

Take i, j such that $D_{i,j}^*$ is minimum

Compute $\Delta_{i,j}, \delta_i, \delta_j$

“Refine” tree (initially start with star-tree)

$D \leftarrow$ “adjusted” D , that is, i -th and j -th column/row are combined into new m -th column/row with entries $D_{k,m} = D_{m,k} = \frac{D_{i,k} + D_{j,k} - D_{i,j}}{2}$

$\forall k \neq i, j, m$

Call Neighbor-Joining(D)

Exmpl - whiteboard

Q: Does this always work in such a perfect way?

A: No, it depends on the distances D !

Algorithm:

Neighbor-Joining(D)

If (D is 1×1 matrix) then stop

Construct D^* from D

Take i, j such that $D_{i,j}^*$ is minimum

Compute $\Delta_{i,j}, \delta_i, \delta_j$

“Refine” tree (initially start with star-tree)

$D \leftarrow$ “adjusted” D , that is, i -th and j -th column/row are combined into new m -th column/row with entries $D_{k,m} = D_{m,k} = \frac{D_{i,k} + D_{j,k} - D_{i,j}}{2}$

$\forall k \neq i, j, m$

Call Neighbor-Joining(D)

Exmpl - whiteboard

Q: Does this always work in such a perfect way?

A: No, it depends on the distances D !

NJ and additive metrics

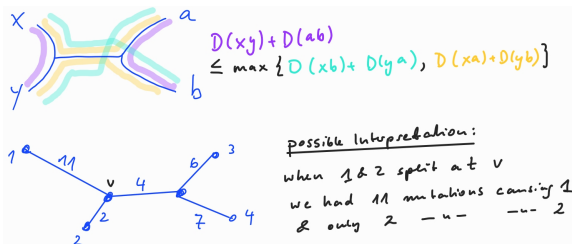
$D: X \times X \rightarrow \mathbb{R}_{\geq 0}$ is an **additive metric** if for all $x, y, a, b \in X$ it holds that

(1) $D(x, y) = 0 \iff x = y$

(2) $D(x, y) = D(y, x)$

(3) $D(x, y) + D(a, b) \leq \max\{D(x, a) + D(y, b), D(x, b) + D(y, a)\}$

Intuition:



A tree T with branch-length δ is additive for matrix D if

$$\text{dist}_T(i, j) = \sum_{\text{edges along unique path connecting } i, j} \delta(i, j) = D_{i, j} \text{ for all leaves } i, j$$

Theorem

There is an additive tree (T, δ) that represents D if and only if D is an additive metric. [without proof]

NJ and additive metrics

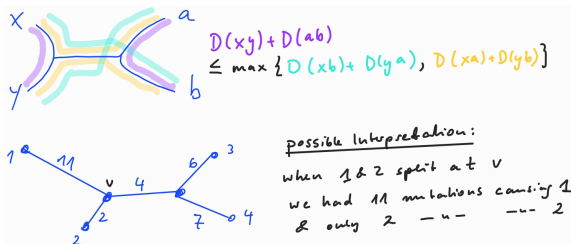
$D: X \times X \rightarrow \mathbb{R}_{\geq 0}$ is an **additive metric** if for all $x, y, a, b \in X$ it holds that

(1) $D(x, y) = 0 \iff x = y$

(2) $D(x, y) = D(y, x)$

(3) $D(x, y) + D(a, b) \leq \max\{D(x, a) + D(y, b), D(x, b) + D(y, a)\}$

Intuition:



A tree T with branch-length δ is additive for matrix D if

$$\text{dist}_T(i, j) = \sum_{\text{edges along unique path connecting } i, j} \delta(i, j) = D_{i,j} \text{ for all leaves } i, j$$

Theorem

There is an additive tree (T, δ) that represents D if and only if D is an additive metric. [without proof]

NJ is based on the concept of minimum-evolution, i.e., the resulting tree will have minimum total branch length.

If D is an additive metric, NJ computes a tree (T, δ) that represents D .

The correctness of the output tree topology is even guaranteed as long as the distance matrix is 'nearly additive', specifically if each entry in the distance matrix differs from the true distance by less than half of the shortest branch length in the tree

Although quite fast, it has a drawback: NJ often assigns negative length to some of the branches.

Summary: Distance-Based Methods

- ▶ Distance-based Methods work well on near-additive or ultrametric data
- ▶ The latter is often violated, however, these methods are quite useful as heuristics
- ▶ We examined two fundamental approaches, but plenty of other methods exist

Observation:

When we use sequence-alignments then we can obtain distances and use Distance-based Methods to compute a tree even with branch-length.

BUT: we loose all information about possible ancestral states!

⇒ other methods ?




Character-Based Methods

Before the “Era of DNA” half a century ago, researchers constructed trees from anatomical/physiological properties called **characters**.

Character-Based Methods

Before the “Era of DNA” half a century ago, researchers constructed trees from anatomical/physiological properties called **characters**.

Example:

	wings	nr of legs
	winged stick-insect yes	6
	wing-less stick-insect no	6
	giant centipede no	42

Character-Based Methods

Character-Based Phylogeny Problem: *Reconstruct a phylogeny from characters*

- ▶ **Input:** An $n \times m$ character table for n taxa and m characters
- ▶ **Output:** A tree in which taxa with similar character values occur near each other

This is by-far not a precise mathematical definition, but it reflects the idea very well

Character-Based Methods

Character-Based Phylogeny Problem: *Reconstruct a phylogeny from characters*

- ▶ **Input:** An $n \times m$ character table for n taxa and m characters
- ▶ **Output:** A tree in which taxa with similar character values occur near each other

This is by-far not a precise mathematical definition, but it reflects the idea very well

Character-Based Methods

Character-Based Phylogeny Problem: *Reconstruct a phylogeny from characters*

- ▶ **Input:** An $n \times m$ character table for n taxa and m characters
- ▶ **Output:** A tree in which taxa with similar character values occur near each other

This is by-far not a precise mathematical definition, but it reflects the idea very well

Example:



	wings	nr of legs
winged stick-insect	yes	6
wing-less stick-insect	no	6

Character-Based Methods

Character-Based Phylogeny Problem: *Reconstruct a phylogeny from characters*

- ▶ **Input:** An $n \times m$ character table for n taxa and m characters
- ▶ **Output:** A tree in which taxa with similar character values occur near each other

This is by-far not a precise mathematical definition, but it reflects the idea very well

Example:



	wings	nr of legs
winged stick-insect	yes	6
wing-less stick-insect	no	6



Character-Based Methods



	wings	nr of legs
winged stick-insect	yes	6
wing-less stick-insect	no	6

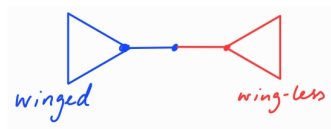


This is quite reasonable and is in line with **Dollo's law of irreversibility** (1893): *Evolution doesn't reinvent the same organ (e.g. wings)*

Let's have a look to the currently best-approximated phylogeny of stick-insects



	wings	nr of legs
winged stick-insect	yes	6
wing-less stick-insect	no	6

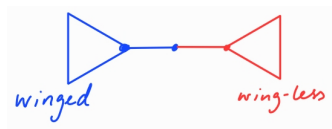


This is quite reasonable and is in line with **Dollo's law of irreversibility** (1893): *Evolution doesn't reinvent the same organ (e.g. wings)*

Let's have a look to the currently best-approximated phylogeny of stick-insects



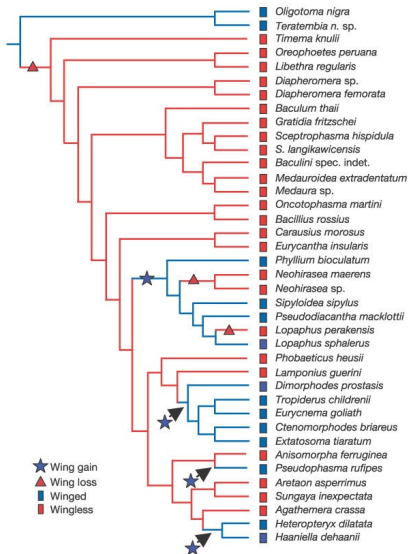
	wings	nr of legs
winged stick-insect	yes	6
wing-less stick-insect	no	6



This is quite reasonable and is in line with **Dollo's law of irreversibility** (1893): *Evolution doesn't reinvent the same organ (e.g. wings)*

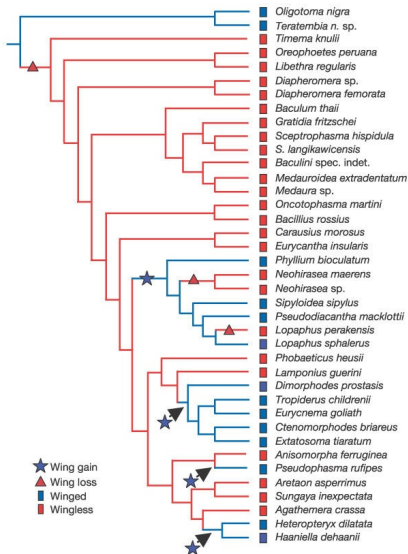
Let's have a look to the currently best-approximated phylogeny of stick-insects

Evolutionary History of Stick-Insects:



What can you observe?

Evolutionary History of Stick-Insects:

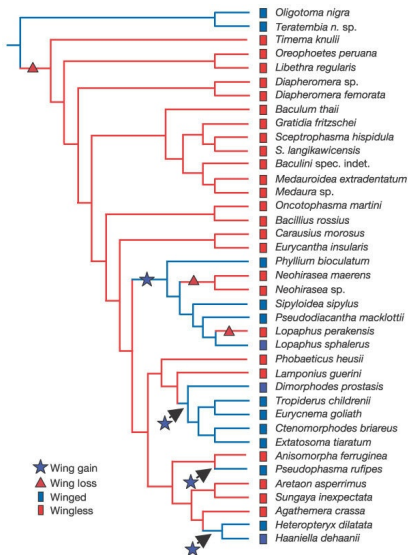


Wings were gained or lost
7times in stick-insects alone!

What happened?

What can you observe?

Evolutionary History of Stick-Insects:



Wings were gained or lost 7 times in stick-insects alone!

What happened?

Evolution did not reinvent wings from scratch!

The genetic information of having wings is not lost, but “suppressed” and “switched-on/off” which can be justified by examining the genomes.

What can you observe?

Character-Based Methods

We do not consider here morphological features as characters but genetic data.

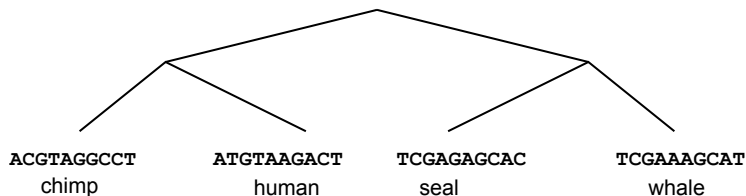
SPECIES	ALIGNMENT	
Chimp	ACGTAGGCCT	} n species
Human	ATGTAAGACT	
Seal	TCGAGAGCAC	
Whale	TCGAAAGCAT	
		
	m characters	

Character-Based Methods

We do not consider here morphological features as characters but genetic data.

SECIES	ALIGNMENT	
Chimp	ACGTAGGCCT	} n species
Human	ATGTAAGACT	
Seal	TCGAGAGCAC	
Whale	TCGAAAGCAT	

} m characters



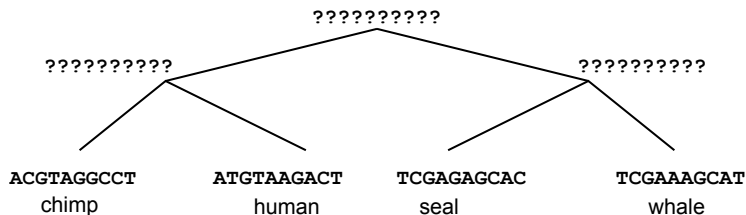
Character-Based Methods

We do not consider here morphological features as characters but genetic data.

SECIES	ALIGNMENT
Chimp	ACGTAGGCCT
Human	ATGTAAGACT
Seal	TCGAGAGCAC
Whale	TCGAAAGCAT

} n species

⏟
 m characters

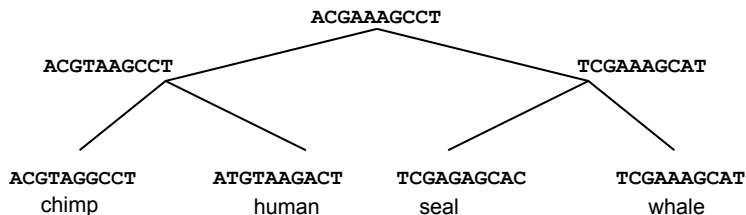


Character-Based Methods

We do not consider here morphological features as characters but genetic data.

SECIES	ALIGNMENT	
Chimp	ACGTAGGCCT	} n species
Human	ATGTAAGACT	
Seal	TCGAGAGCAC	
Whale	TCGAAAGCAT	

} m characters



Let us assign sequences to T

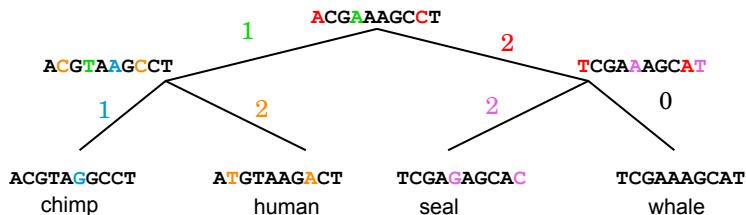
Character-Based Methods

We do not consider here morphological features as characters but genetic data.

SECIES	ALIGNMENT
Chimp	ACGTAGGCCT
Human	ATGTAAGACT
Seal	TCGAGAGCAC
Whale	TCGAAAGCAT

} n species

⏟
 m characters

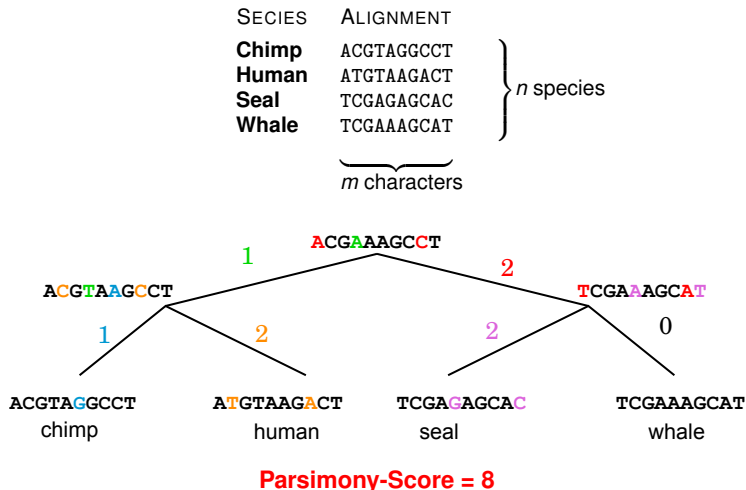


Let us assign sequences to T

Parsimony-Score: Sum of Hamming-distances along edges in T

Character-Based Methods

We do not consider here morphological features as characters but genetic data.



Parsimony-Score: Sum of Hamming-distances along edges in T

Small Parsimony Problem

= Find the most parsimonious labeling of internal nodes of given tree.

In: A rooted tree T whose leaves are labeled by a string of length m

Out: A labeling of all inner vertices by strings of length m that minimizes the tree's parsimony score

Assuming the columns of a multiple alignment are independent from each other we can simplify the problem as follows:

In: A rooted tree T whose leaves are labeled by a **single symbol**

Out: A labeling of all inner vertices by **single symbol** that minimizes the tree's parsimony score

And repeat the latter for each of the m columns.

Q: Why Parsimony?

A: Ocham's razor (1347) *The simplest explanation is usually the best one.*
[in a very simplified version]

Small Parsimony Problem

= Find the most parsimonious labeling of internal nodes of given tree.

In: A rooted tree T whose leaves are labeled by a string of length m

Out: A labeling of all inner vertices by strings of length m that minimizes the tree's parsimony score

Assuming the columns of a multiple alignment are independent from each other we can simplify the problem as follows:

In: A rooted tree T whose leaves are labeled by a **single symbol**

Out: A labeling of all inner vertices by **single symbol** that minimizes the tree's parsimony score

And repeat the latter for each of the m columns.

Q: Why Parsimony?

A: Ocham's razor (1347) *The simplest explanation is usually the best one.*
[in a very simplified version]

Small Parsimony Problem

= Find the most parsimonious labeling of internal nodes of given tree.

In: A rooted tree T whose leaves are labeled by a string of length m

Out: A labeling of all inner vertices by strings of length m that minimizes the tree's parsimony score

Assuming the columns of a multiple alignment are independent from each other we can simplify the problem as follows:

In: A rooted tree T whose leaves are labeled by a **single symbol**

Out: A labeling of all inner vertices by **single symbol** that minimizes the tree's parsimony score

And repeat the latter for each of the m columns.

Q: Why Parsimony?

A: Ocham's razor (1347) *The simplest explanation is usually the best one.*
[in a very simplified version]

Small Parsimony Problem

= Find the most parsimonious labeling of internal nodes of given tree.

In: A rooted tree T whose leaves are labeled by a string of length m

Out: A labeling of all inner vertices by strings of length m that minimizes the tree's parsimony score

Assuming the columns of a multiple alignment are independent from each other we can simplify the problem as follows:

In: A rooted tree T whose leaves are labeled by a **single symbol**

Out: A labeling of all inner vertices by **single symbol** that minimizes the tree's parsimony score

And repeat the latter for each of the m columns.

Q: Why Parsimony?

A: Ocham's razor (1347) *The simplest explanation is usually the best one.*
[in a very simplified version]

Given a binary tree with leaves labeled by a symbol.

We traverse the tree from the leaves to the root such that when a vertex is processed, all its children have already been processed. Obviously the root is the last node processed by this traversal. During this phase, we collect putative states for the labeling of each vertex v , stored in a candidate set X_v .

In (1b), Assume an inner vertex v with children u and w . If u and w share common candidates, these are candidates for v as well. Otherwise, the candidates of both children have to be considered as candidates for v .

1. (Bottom-up phase)

1a. (Leaves) for each leaf ℓ , set $X_\ell = \{\text{label of } \ell\}$

1b. (Inner vertices)

$$X_v = \begin{cases} X_u \cap X_w, & \text{if } X_u \cap X_w \neq \emptyset \\ X_u \cup X_w, & \text{otherwise} \end{cases}$$

The most parsimonious reconstruction of character-states (symbols) at the inner vertices is then obtained in a top-down pass according to the following rules:

2. (Top-down refinement)

2a. (Root) If the candidate set of the root contains more than one element, arbitrarily assign one of these symbols to the root.

2b. (Other vertices) Let v be a child of node u , and let a denote the symbol assigned to u .

If a is contained in X_v , assign it to node v as well.

Otherwise, arbitrarily assign any state from X_v to node v .

For the proof of correctness we refer to "*Hartigan, Minimum mutation fits to a given tree. Biometrics, 1973*" were a generalized version of this algorithm is studied that also deals with non-binary tree and to find all co-optimal solutions.

Given a binary tree with leaves labeled by a symbol.

We traverse the tree from the leaves to the root such that when a vertex is processed, all its children have already been processed. Obviously the root is the last node processed by this traversal. During this phase, we collect putative states for the labeling of each vertex v , stored in a candidate set X_v .

In (1b), Assume an inner vertex v with children u and w . If u and w share common candidates, these are candidates for v as well. Otherwise, the candidates of both children have to be considered as candidates for v .

1. (Bottom-up phase)

1a. (Leaves) for each leaf ℓ , set $X_\ell = \{\text{label of } \ell\}$

1b. (Inner vertices)

$$X_v = \begin{cases} X_u \cap X_w, & \text{if } X_u \cap X_w \neq \emptyset \\ X_u \cup X_w, & \text{otherwise} \end{cases}$$

The most parsimonious reconstruction of character-states (symbols) at the inner vertices is then obtained in a top-down pass according to the following rules:

2. (Top-down refinement)

2a. (Root) If the candidate set of the root contains more than one element, arbitrarily assign one of these symbols to the root.

2b. (Other vertices) Let v be a child of node u , and let a denote the symbol assigned to u .

If a is contained in X_v , assign it to node v as well.

Otherwise, arbitrarily assign any state from X_v to node v .

For the proof of correctness we refer to "*Hartigan, Minimum mutation fits to a given tree. Biometrics, 1973*" were a generalized version of this algorithm is studied that also deals with non-binary tree and to find all co-optimal solutions.

Given a binary tree with leaves labeled by a symbol.

We traverse the tree from the leaves to the root such that when a vertex is processed, all its children have already been processed. Obviously the root is the last node processed by this traversal. During this phase, we collect putative states for the labeling of each vertex v , stored in a candidate set X_v .

In (1b), Assume an inner vertex v with children u and w . If u and w share common candidates, these are candidates for v as well. Otherwise, the candidates of both children have to be considered as candidates for v .

1. (Bottom-up phase)

1a. (Leaves) for each leaf ℓ , set $X_\ell = \{\text{label of } \ell\}$

1b. (Inner vertices)

$$X_v = \begin{cases} X_u \cap X_w, & \text{if } X_u \cap X_w \neq \emptyset \\ X_u \cup X_w, & \text{otherwise} \end{cases}$$

The most parsimonious reconstruction of character-states (symbols) at the inner vertices is then obtained in a top-down pass according to the following rules:

2. (Top-down refinement)

2a. (Root) If the candidate set of the root contains more than one element, arbitrarily assign one of these symbols to the root.

2b. (Other vertices) Let v be a child of node u , and let a denote the symbol assigned to u .

If a is contained in X_v , assign it to node v as well.

Otherwise, arbitrarily assign any state from X_v to node v .

For the proof of correctness we refer to "*Hartigan, Minimum mutation fits to a given tree. Biometrics, 1973*" were a generalized version of this algorithm is studied that also deals with non-binary tree and to find all co-optimal solutions.

Given a binary tree with leaves labeled by a symbol.

We traverse the tree from the leaves to the root such that when a vertex is processed, all its children have already been processed. Obviously the root is the last node processed by this traversal. During this phase, we collect putative states for the labeling of each vertex v , stored in a candidate set X_v .

In (1b), Assume an inner vertex v with children u and w . If u and w share common candidates, these are candidates for v as well. Otherwise, the candidates of both children have to be considered as candidates for v .

1. (Bottom-up phase)

1a. (Leaves) for each leaf ℓ , set $X_\ell = \{\text{label of } \ell\}$

1b. (Inner vertices)

$$X_v = \begin{cases} X_u \cap X_w, & \text{if } X_u \cap X_w \neq \emptyset \\ X_u \cup X_w, & \text{otherwise} \end{cases}$$

The most parsimonious reconstruction of character-states (symbols) at the inner vertices is then obtained in a top-down pass according to the following rules:

2. (Top-down refinement)

2a. (Root) If the candidate set of the root contains more than one element, arbitrarily assign one of these symbols to the root.

2b. (Other vertices) Let v be a child of node u , and let a denote the symbol assigned to u .

If a is contained in X_v , assign it to node v as well.

Otherwise, arbitrarily assign any state from X_v to node v .

For the proof of correctness we refer to "*Hartigan, Minimum mutation fits to a given tree. Biometrics, 1973*" were a generalized version of this algorithm is studied that also deals with non-binary tree and to find all co-optimal solutions.

Fitch Algorithm (Walther M. Fitch, 1971)

1a. (Leaves) for each leaf ℓ , set $X_\ell = \{\text{label of } \ell\}$

1b. (Inner vertices)

$$X_v = \begin{cases} X_u \cap X_w, & \text{if } X_u \cap X_w \neq \emptyset \\ X_u \cup X_w, & \text{otherwise} \end{cases}$$

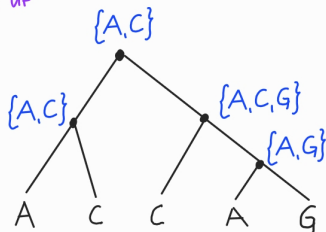
2a. (Root) If the candidate set of the root contains more than one element, arbitrarily assign one of these symbols to the root.

2b. (Other vertices) Let v be a child of node u , and let a denote the symbol assigned to u .

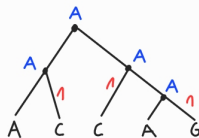
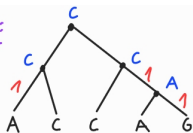
If a is contained in X_v , assign it to node v as well.

Otherwise, arbitrarily assign any state from X_v to node v .

(1) BOTTOM UP



(2) TOP-DOWN:
(possible solutions)



Given a (not necessarily binary) tree with leaves labeled by a symbol.

The tree is traversed bottom-up. During this traversal, assume we process a vertex u . Define $s(u)$ as the cost of the min. pars.-score for the subtree $T(u)$ rooted at u . Let $s_a(u)$ be the cost of the best labeling of $T(u)$ when u is required to be labeled with symbol a . Obviously, $s(u) = \min_a s_a(u)$.

1. (Bottom-up phase)

1a. (Leaves)

The symbol for each leaf ℓ is fixed and we put $s_a(\ell) = 0$ if label of ℓ is a and, otherwise, $s_a(\ell) = \infty$

1b. (Inner vertices)

The recurrence relation to compute $s_a(u)$ for inner vertex u is given by

$$s_a(u) = \sum_{\text{child } v \text{ of } u} \min_{\text{all symbols } b} (s_b(v) + \mathbb{1}_{a,b})$$

2. (Top-down refinement) The optimal assignment of states to the internal nodes is then obtained in a backtracing phase.

2a. (Root)

The root ρ is assigned a state a such that $s(\rho) = s_a(\rho)$.

2b. (Other vertices)

In a top-down traversal, the child v of an already labeled vertex u (say, u was labeled with state a) is assigned a state b that yielded the minimum in the bottom-up pass, i.e., where

$$\mathbb{1}_{a,b} + s_b(v) = \min_{b'} (\mathbb{1}_{a,b'} + s_{b'}(v))$$

Correctness is exercise, but follows essential from the fact that we consider all possibilities.

Sankoff Algorithm (David Sankoff, 1971)

1a. (Leaves) The symbol for each leaf ℓ is fixed and we put $s_a(\ell) = 0$ if label of ℓ is a and, otherwise, $s_a(\ell) = \infty$

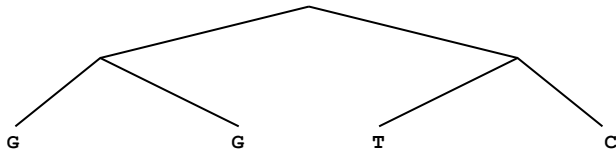
1b. (Inner vertices) The recurrence relation to compute $s_a(u)$ for inner vertex u is given by

$$s_a(u) = \sum_{\text{child } v \text{ of } u} \min_{\text{all symbols } b} (s_b(v) + \mathbb{1}_{a,b})$$

2a. (Root) The root ρ is assigned a state a such that $s(\rho) = s_a(\rho)$.

2b. (Other vertices) In a top-down traversal, the child v of an already labeled vertex u (say, u was labeled with state a) is assigned a state b that yielded the minimum in the bottom-up pass, i.e., where

$$\mathbb{1}_{a,b} + s_b(v) = \min_{b'} (\mathbb{1}_{a,b'} + s_{b'}(v))$$



Sankoff Algorithm (David Sankoff, 1971)

1a. (Leaves) The symbol for each leaf ℓ is fixed and we put $s_a(\ell) = 0$ if label of ℓ is a and, otherwise, $s_a(\ell) = \infty$

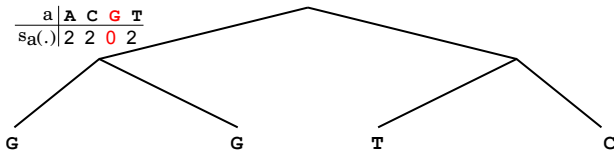
1b. (Inner vertices) The recurrence relation to compute $s_a(u)$ for inner vertex u is given by

$$s_a(u) = \sum_{\text{child } v \text{ of } u} \min_{\text{all symbols } b} (s_b(v) + \mathbb{1}_{a,b})$$

2a. (Root) The root ρ is assigned a state a such that $s(\rho) = s_a(\rho)$.

2b. (Other vertices) In a top-down traversal, the child v of an already labeled vertex u (say, u was labeled with state a) is assigned a state b that yielded the minimum in the bottom-up pass, i.e., where

$$\mathbb{1}_{a,b} + s_b(v) = \min_{b'} (\mathbb{1}_{a,b'} + s_{b'}(v))$$



Sankoff Algorithm (David Sankoff, 1971)

1a. (Leaves) The symbol for each leaf ℓ is fixed and we put $s_a(\ell) = 0$ if label of ℓ is a and, otherwise, $s_a(\ell) = \infty$

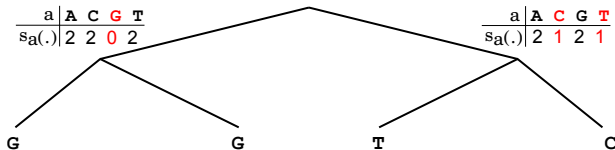
1b. (Inner vertices) The recurrence relation to compute $s_a(u)$ for inner vertex u is given by

$$s_a(u) = \sum_{\text{child } v \text{ of } u} \min_{\text{all symbols } b} (s_b(v) + \mathbb{1}_{a,b})$$

2a. (Root) The root ρ is assigned a state a such that $s(\rho) = s_a(\rho)$.

2b. (Other vertices) In a top-down traversal, the child v of an already labeled vertex u (say, u was labeled with state a) is assigned a state b that yielded the minimum in the bottom-up pass, i.e., where

$$\mathbb{1}_{a,b} + s_b(v) = \min_{b'} (\mathbb{1}_{a,b'} + s_{b'}(v))$$



Sankoff Algorithm (David Sankoff, 1971)

1a. (Leaves) The symbol for each leaf ℓ is fixed and we put $s_a(\ell) = 0$ if label of ℓ is a and, otherwise, $s_a(\ell) = \infty$

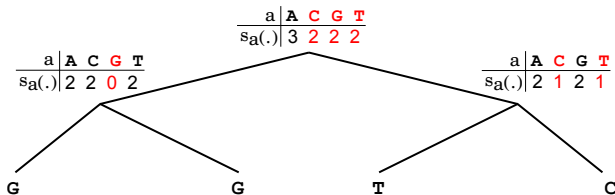
1b. (Inner vertices) The recurrence relation to compute $s_a(u)$ for inner vertex u is given by

$$s_a(u) = \sum_{\text{child } v \text{ of } u} \min_{\text{all symbols } b} (s_b(v) + \mathbb{1}_{a,b})$$

2a. (Root) The root ρ is assigned a state a such that $s(\rho) = s_a(\rho)$.

2b. (Other vertices) In a top-down traversal, the child v of an already labeled vertex u (say, u was labeled with state a) is assigned a state b that yielded the minimum in the bottom-up pass, i.e., where

$$\mathbb{1}_{a,b} + s_b(v) = \min_{b'} (\mathbb{1}_{a,b'} + s_{b'}(v))$$



Sankoff Algorithm (David Sankoff, 1971)

1a. (Leaves) The symbol for each leaf ℓ is fixed and we put $s_a(\ell) = 0$ if label of ℓ is a and, otherwise, $s_a(\ell) = \infty$

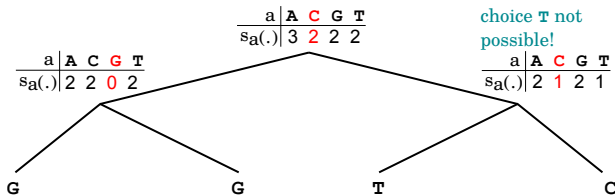
1b. (Inner vertices) The recurrence relation to compute $s_a(u)$ for inner vertex u is given by

$$s_a(u) = \sum_{\text{child } v \text{ of } u} \min_{\text{all symbols } b} (s_b(v) + \mathbb{1}_{a,b})$$

2a. (Root) The root ρ is assigned a state a such that $s(\rho) = s_a(\rho)$.

2b. (Other vertices) In a top-down traversal, the child v of an already labeled vertex u (say, u was labeled with state a) is assigned a state b that yielded the minimum in the bottom-up pass, i.e., where

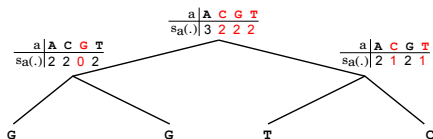
$$\mathbb{1}_{a,b} + s_b(v) = \min_{b'} (\mathbb{1}_{a,b'} + s_{b'}(v))$$



Summary Fitch- and Sankoff-Algorithm

Both solve the small parsimony problem and run in polynomial-time.

The red-colored vertices are precisely the sets X_v computed with the Fitch alg.:



But Fitch is restricted to “unit costs” $\mathbb{1}_{a,b}$ which can be replaced in Sankoff's alg. by an arbitrary cost function.

In Sankoff's alg. backtracking can be used to obtain all optimal solutions and it works on non-binary trees.

Large Parsimony Problem

In: n strings of length m

Out: Find a tree on n leaves together with a labeling of all inner vertices by strings of length m that minimizes the tree's parsimony score

This problem is NP-hard!

⇒ heuristics are needed [not part of this lecture]

Consensus-Based Methods

A simple example:

Assume we have partial information about similarities about between some taxa A, B, C, D, E such as

- ▶ A and B are closer related than A to C and B to C
- ▶ C and D are closer related than C to E and D to E

A simple example:

Assume we have partial information about similarities about between some taxa A, B, C, D, E such as

- ▶ A and B are closer related than A to C and B to C
- ▶ C and D are closer related than C to E and D to E

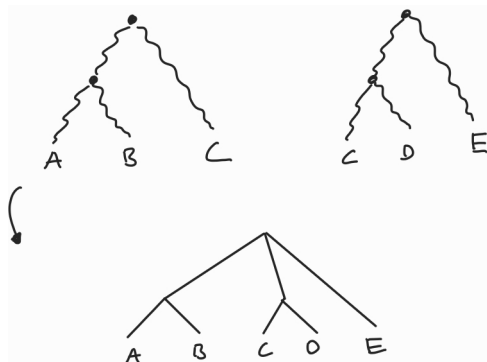


Is there a rooted tree that reflects both observations?

A simple example:

Assume we have partial information about similarities about between some taxa A, B, C, D, E such as

- ▶ A and B are closer related than A to C and B to C
- ▶ C and D are closer related than C to E and D to E



Is there a rooted tree that reflects both observations?

Central Idea:

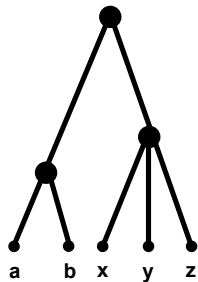
- ▶ Find a consensus tree that reflects all partial information as “best as possible”.

Motivation:

- ▶ Combine many trees constructed from different data sets.
- ▶ Computationally expensive methods may yield highly accurate trees for small, overlapping subsets of the objects.
- ▶ Most individual studies investigate relatively few species. Supertrees allow us to deduce new evolutionary relationships.

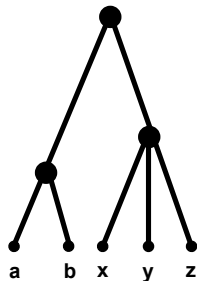
whiteboard: DEF rooted triple, displayed, compatible

Rooted tree T:



connected, acyclic
graph

Rooted tree T:

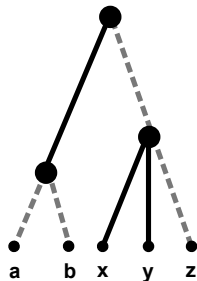


Triples:

T **displays** a triple $ab|z$ if the path from a to b does not intersect the path from z to the root.

$$\iff \text{lca}_T(a, b) \prec_T \text{lca}_T(a, c) = \text{lca}_T(b, c)$$

Rooted tree T:

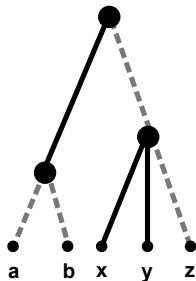


Triples:

T **displays** a triple $ab|z$ if the path from a to b does not intersect the path from z to the root.

$$\iff \text{lca}_T(a, b) \prec_T \text{lca}_T(a, c) = \text{lca}_T(b, c)$$

Rooted tree T:



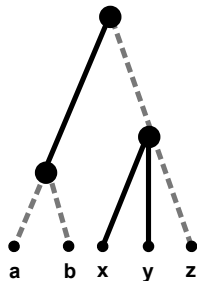
Triples:

T **displays** a triple $ab|z$ if the path from a to b does not intersect the path from z to the root.

$$\iff \text{lca}_T(a, b) \prec_T \text{lca}_T(a, c) = \text{lca}_T(b, c)$$

$$\mathcal{R}(T) = \{ab|x, ab|y, ab|z, xy|a \dots\}$$

Rooted tree T :



Triples:

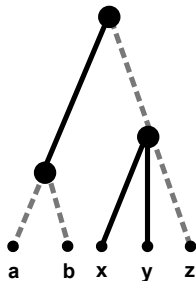
T **displays** a triple $ab|z$ if the path from a to b does not intersect the path from z to the root.

$$\iff \text{lca}_T(a, b) \prec_T \text{lca}_T(a, c) = \text{lca}_T(b, c)$$

$$\mathcal{R}(T) = \{ab|x, ab|y, ab|z, xy|a \dots\}$$

For a set R of triples let $L(R) := \cup_{xy|z \in R} \{x, y, z\}$.

Rooted tree T:



Triples:

T **displays** a triple $ab|z$ if the path from a to b does not intersect the path from z to the root.

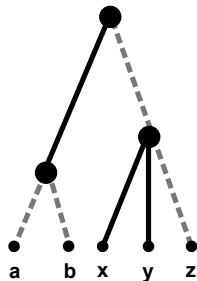
$$\iff \text{lca}_T(a, b) \prec_T \text{lca}_T(a, c) = \text{lca}_T(b, c)$$

$$\mathcal{R}(T) = \{ab|x, ab|y, ab|z, xy|a \dots\}$$

For a set R of triples let $L(R) := \cup_{xy|z \in R} \{x, y, z\}$.

An arbitrary set R of triples is **compatible**, if there is a tree T on $L(R)$ with $R \subseteq \mathcal{R}(T)$

Rooted tree T:



Triples:

T **displays** a triple $ab|z$ if the path from a to b does not intersect the path from z to the root.

$$\iff \text{lca}_T(a, b) \prec_T \text{lca}_T(a, c) = \text{lca}_T(b, c)$$

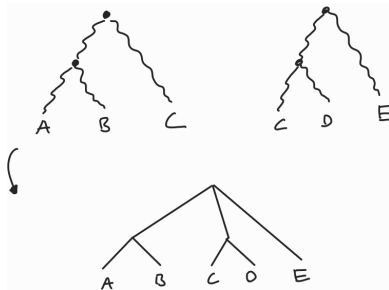
$$\mathcal{R}(T) = \{ab|x, ab|y, ab|z, xy|a \dots\}$$

For a set R of triples let $L(R) := \cup_{xy|z \in R} \{x, y, z\}$.

An arbitrary set R of triples is **compatible**, if there is a tree T on $L(R)$ with $R \subseteq \mathcal{R}(T)$

When is a set R of triples compatible?

$R = \{AB|C, CD|E\}$ is compatible:



$R = \{AB|C, CB|A\}$ is not compatible.

How to test compatibility of R ?

Observation: Assume there is a rooted tree T that displays R .

If $xy|z \in R$ then x and y cannot be descendants of two distinct children of the root ρ_T

Central Idea:

- ▶ Determine for potential tree T on $L(R)$ for R the set of leaves that are descendants of children of the root.

Hence, we want to find a partition X_1, \dots, X_ℓ of $L(R)$:

Note $xy|z \in R$ implies that $x, y \in X_i$ for some $i \in \{1, \dots, \ell\}$

- ▶ Then recurse on each such child.

Rooted Trees, Triples and Compatibility

How to test compatibility of R ?

Observation: Assume there is a rooted tree T that displays R .

If $xy|z \in R$ then x and y cannot be descendants of two distinct children of the root ρ_T



Central Idea:

- ▶ Determine for potential tree T on $L(R)$ for R the set of leaves that are descendants of children of the root.

Hence, we want to find a partition X_1, \dots, X_ℓ of $L(R)$:

Note $xy|z \in R$ implies that $x, y \in X_i$ for some $i \in \{1, \dots, \ell\}$

- ▶ Then recurse on each such child.

Rooted Trees, Triples and Compatibility

How to test compatibility of R ?

Observation: Assume there is a rooted tree T that displays R .

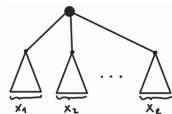
If $xy|z \in R$ then x and y cannot be descendants of two distinct children of the root ρ_T



Central Idea:

- Determine for potential tree T on $L(R)$ for R the set of leaves that are descendants of children of the root.

Hence, we want to find a partition X_1, \dots, X_ℓ of $L(R)$:



Note $xy|z \in R$ implies that $x, y \in X_i$ for some $i \in \{1, \dots, \ell\}$

- Then recurse on each such child.

Rooted Trees, Triples and Compatibility

Define for a set R of triples and a leaf set L the set

$$R|_L := \{xy|z \in R: x, y, z \in L\}$$

Example:

$$R = \{ab|c, ab|d, ax|y\}$$

$$R|_L = \{ab|c, ax|y\} \text{ for } L = \{a, b, c, x, y\}$$

$$R|_L = \emptyset \text{ for e.g. } L = \{a, b, y\}$$

Comparative graph $G[R,L]$:

Given set R of triples and a leaf set L .

Then $G[R, L]$ has vertex set L and $\{x, y\}$ is an edge iff $\exists xy|z \in R|_L$

Rooted Trees, Triples and Compatibility

Define for a set R of triples and a leaf set L the set

$$R|_L := \{xy|z \in R: x, y, z \in L\}$$

Example:

$$R = \{ab|c, ab|d, ax|y\}$$

$$R|_L = \{ab|c, ax|y\} \text{ for } L = \{a, b, c, x, y\}$$

$$R|_L = \emptyset \text{ for e.g. } L = \{a, b, y\}$$

Comparative graph $G[R,L]$:

Given set R of triples and a leaf set L .

Then $G[R, L]$ has vertex set L and $\{x, y\}$ is an edge iff $\exists xy|z \in R|_L$

Rooted Trees, Triples and Compatibility

Define for a set R of triples and a leaf set L the set

$$R|_L := \{xy|z \in R: x, y, z \in L\}$$

Example:

$$R = \{ab|c, ab|d, ax|y\}$$

$$R|_L = \{ab|c, ax|y\} \text{ for } L = \{a, b, c, x, y\}$$

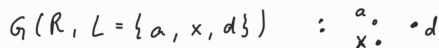
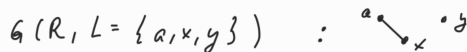
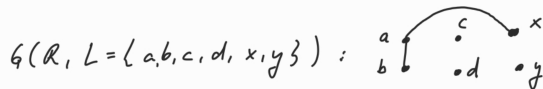
$$R|_L = \emptyset \text{ for e.g. } L = \{a, b, y\}$$

Comparative graph $G[R, L]$:

Given set R of triples and a leaf set L .

Then $G[R, L]$ has vertex set L and $\{x, y\}$ is an edge iff $\exists xy|z \in R|_L$

$$R = \{ ab|c, ab|d, ax|y \}$$



Rooted Trees, Triples and Compatibility

Is $R = \{ab|c, ab|d, ax|y\}$ compatible? Need to find tree T that displays R

Hence, we want to find a partition X_1, \dots, X_ℓ of $L(R)$:

Note $xy|z \in R$ implies that $x, y \in X_i$ for some $i \in \{1, \dots, \ell\}$

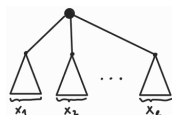
IDEA: connect x and y as an edge and look at connected components!

This is precisely what is reflected by the Comparative graph $G[R, L]$!

Rooted Trees, Triples and Compatibility

Is $R = \{ab|c, ab|d, ax|y\}$ compatible? Need to find tree T that displays R

Hence, we want to find a partition X_1, \dots, X_ℓ of $L(R)$:



Note $xy|z \in R$ implies that $x, y \in X_i$ for some $i \in \{1, \dots, \ell\}$

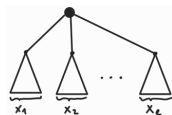
IDEA: connect x and y as an edge and look at connected components!

This is precisely what is reflected by the Comparative graph $G[R, L]$!

Rooted Trees, Triples and Compatibility

Is $R = \{ab|c, ab|d, ax|y\}$ compatible? Need to find tree T that displays R

Hence, we want to find a partition X_1, \dots, X_ℓ of $L(R)$:



Note $xy|z \in R$ implies that $x, y \in X_i$ for some $i \in \{1, \dots, \ell\}$

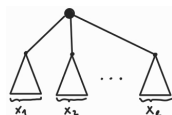
IDEA: connect x and y as an edge and look at connected components!

This is precisely what is reflected by the Comparative graph $G[R, L]$!

Rooted Trees, Triples and Compatibility

Is $R = \{ab|c, ab|d, ax|y\}$ compatible? Need to find tree T that displays R

Hence, we want to find a partition X_1, \dots, X_ℓ of $L(R)$:



Note $xy|z \in R$ implies that $x, y \in X_i$ for some $i \in \{1, \dots, \ell\}$

IDEA: connect x and y as an edge and look at connected components!

This is precisely what is reflected by the Comparative graph $G[R, L]$!

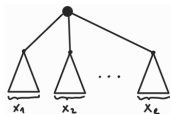
$$R = \{ax|b, ab|c, cd|y\}$$



Rooted Trees, Triples and Compatibility

Is $R = \{ab|c, ab|d, ax|y\}$ compatible? Need to find tree T that displays R

Hence, we want to find a partition X_1, \dots, X_ℓ of $L(R)$:

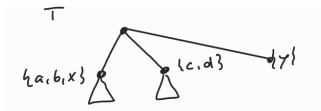


Note $xy|z \in R$ implies that $x, y \in X_i$ for some $i \in \{1, \dots, \ell\}$

IDEA: connect x and y as an edge and look at connected components!

This is precisely what is reflected by the Comparative graph $G[R, L]$!

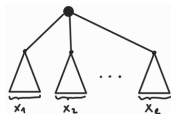
$$R = \{ax|b, ab|c, cd|y\}$$



Rooted Trees, Triples and Compatibility

Is $R = \{ab|c, ab|d, ax|y\}$ compatible? Need to find tree T that displays R

Hence, we want to find a partition X_1, \dots, X_ℓ of $L(R)$:

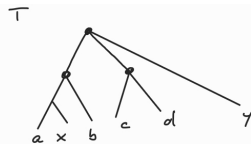


Note $xy|z \in R$ implies that $x, y \in X_i$ for some $i \in \{1, \dots, \ell\}$

IDEA: connect x and y as an edge and look at connected components!

This is precisely what is reflected by the Comparative graph $G[R, L]$!

$$R = \{ax|b, ab|c, cd|y\}$$



by Aho, Sagiv, Szymanski and Ullman (1981)

BUILD(R, v, T, L)

IF($|L| = 1$)

// $L = \{x\}$

output rooted tree $\overset{\bullet}{x}$

IF($|L| = 2$)

// $L = \{x, y\}$

output rooted tree obtained by attaching two vertices to v labelled x and y .

IF($|L| \geq 3$)

Construct $G[R, L]$

Let L_1, \dots, L_k be the vertex set of conn. comp. of $G[R, L]$

IF($k = 1$) **RETURN** "*R not compatible*"

FOR($i = 1, \dots, k$)

call **BUILD**(R, v_i, T_i, L_i)

IF(**BUILD**(R, v_i, T_i, L_i) outputs a tree T_i)

attach T_i to v via edge $\{v, v_i\}$.

Further Examples: Whiteboard

Theorem

BUILD runs in $O(|L||R|)$ -time and is correct
proof sketch: whiteboard

by Aho, Sagiv, Szymanski and Ullman (1981)

BUILD(R, v, T, L)

IF($|L| = 1$)

// $L = \{x\}$

output rooted tree $\overset{\bullet}{x}$

IF($|L| = 2$)

// $L = \{x, y\}$

output rooted tree obtained by attaching two vertices to v labelled x and y .

IF($|L| \geq 3$)

Construct $G[R, L]$

Let L_1, \dots, L_k be the vertex set of conn. comp. of $G[R, L]$

IF($k = 1$) **RETURN** "*R not compatible*"

FOR($i = 1, \dots, k$)

call **BUILD**(R, v_i, T_i, L_i)

IF(**BUILD**(R, v_i, T_i, L_i) outputs a tree T_i)

attach T_i to v via edge $\{v, v_i\}$.

Further Examples: [Whiteboard](#)

Theorem

BUILD runs in $O(|L||R|)$ -time and is correct
proof sketch: [whiteboard](#)

by Aho, Sagiv, Szymanski and Ullman (1981)

BUILD(R, v, T, L)

IF($|L| = 1$)

// $L = \{x\}$

output rooted tree $\overset{\bullet}{x}$

IF($|L| = 2$)

// $L = \{x, y\}$

output rooted tree obtained by attaching two vertices to v labelled x and y .

IF($|L| \geq 3$)

Construct $G[R, L]$

Let L_1, \dots, L_k be the vertex set of conn. comp. of $G[R, L]$

IF($k = 1$) **RETURN** "*R not compatible*"

FOR($i = 1, \dots, k$)

call **BUILD**(R, v_i, T_i, L_i)

IF(**BUILD**(R, v_i, T_i, L_i) outputs a tree T_i)

attach T_i to v via edge $\{v, v_i\}$.

Further Examples: [Whiteboard](#)

Theorem

BUILD runs in $O(|L||R|)$ -time and is correct

proof sketch: whiteboard