Algorithm Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sankoff’s &amp; Fitch’s Alg.</td>
<td>Characters, T</td>
<td>A, B</td>
</tr>
<tr>
<td>Perfect Phylogeny</td>
<td>Characters</td>
<td>A, B, T</td>
</tr>
<tr>
<td>Felsenstein</td>
<td>Characters, T, B</td>
<td>A</td>
</tr>
</tbody>
</table>

T = tree topology
B = branch lengths
A = ancestral states
**Pairwise Compatibility Test**  
*(Wilson 1965)*

Binary characters $i$ and $j$ are pairwise compatible if and only if:

- $j$ is homogenous w.r.t $i_0$ or $i_1$.

Equivalently:

- $i_1$ and $j_1$ are disjoint or one contains the other

Equivalently:

<table>
<thead>
<tr>
<th>$i$</th>
<th>$j$</th>
<th>$k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>A 0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>B 0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>C 1</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>D 1</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
<td>E 0</td>
</tr>
</tbody>
</table>

All 4 rows do **not** exist

$(0,0)$, $(0,1)$, $(1,0)$, $(1,1)$

---

**Pairwise Compatibility Theorem**  
*(Estabrook et al. 1976)*

A set $S$ of binary characters is mutually compatible if and only if all pairs $c$ and $c'$ of characters in $S$ are pairwise compatible.

Pairwise compatibility $\leftrightarrow$ mutual compatibility.
Perfect Phylogeny

A set of mutually compatible binary characters gives a **perfect phylogeny**:

1. **Evolutionary model**
   - Binary characters {0,1}
   - Each character changes state only once in evolutionary history (no homoplasies!).
2. **Tree in which every mutation is on an edge of the tree**.
   - All the species in one sub-tree contain a 0, and all species in the other contain a 1.
   - For simplicity, assume root = (0, 0, 0, 0, 0)

*Last time: algorithm to reconstruct a tree.*

---

Trees and Splits

- Given a set $X$, a **split** is a partition of $X$ into two non-empty subsets $A$ and $B$. $X = A \mid B$.
- For a phylogenetic tree $T$ with leaves $L$, each edge $e$ defines a split $L_e = A \mid B$, where $A$ and $B$ are the leaves in the subtrees obtained by removing $e$.

In perfect phylogeny, edges where binary character changes state gave split $i_0$ and $i_1$.

We will return to splits in a future lecture.
Splits Equivalence Theorem

A phylogenetic tree T defines a collection of splits \( \Sigma(T) = \{ L_e \mid e \text{ is edge in } T \} \).

Splits \( A_1 \mid B_1 \) and \( A_2 \mid B_2 \) are \textit{pairwise compatible} if at least one of \( A_1 \cap A_2 \), \( A_1 \cap B_2 \), \( B_1 \cap A_2 \), and \( B_1 \cap B_2 \) is the empty set.

**Splits Equivalence Theorem**: Let \( \Sigma \) be a collection of splits. There is a phylogenetic tree such that \( \Sigma(T) = \Sigma \) if and only if the splits in \( \Sigma \) are pairwise compatible.

The Pairwise Compatibility Theorem (for binary characters) follows from this theorem.

Outline

Distance-based methods for phylogenetic tree reconstruction.

- Review of distances/metrics.
- Tree distances and additive distances
  - Small and large phylogeny problems.
- Non-additive distances and clustering
  - UPGMA and ultrametric distances.
Distances

A **distance** on a set $X$ is a function $d: X \rightarrow \mathbb{R}$ satisfying:
- $d(x, y) \geq 0$, with equality iff $x = y$.
- For all $x, y \in X$, $d(x, y) = d(y, x)$ [symmetry]
- For all $x, y, z \in X$, $d(x, z) \leq d(x, y) + d(y, z)$ [triangle inequality]

Examples:
- $X = \text{real numbers}$, $d(x, y) = |x - y|$ is distance.
- $X = \text{strings over some alphabet}$, $d_H(s, t) =$ number of positions where $s$ and $t$ differ is called Hamming distance.

Distances in Biological Data

- String distances (e.g. Hamming distance, edit distance) on DNA/protein sequence data
- Substitution model (Jukes-Cantor, Kimura, etc.): scores for particular changes $A \rightarrow T, C \rightarrow G, \text{etc.}$

Rat: $\text{ACAGTCACGCCCCACACGT}$
Mouse: $\text{ACAGTGACCCACACACGT}$
Gorilla: $\text{CCTGTACGTAAACAAACGA}$
Chimpanzee: $\text{CCTGTAGGTAGAAACAGA}$
Human: $\text{CCTGTAGGTAGACACAGA}$
Distance Matrix

• For \( n \) species, form \( n \times n \) distance matrix \( D_{ij} \)
• Example: \( D_{ij} = \) edit distance between a gene in species \( i \) and species \( j \).

<table>
<thead>
<tr>
<th></th>
<th>Mouse:</th>
<th>Gorilla:</th>
<th>Chimpanzee:</th>
<th>Human:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ACATGACGCCACACACCT</td>
<td>CCTGCGACGTAAACAAACG</td>
<td>CCTGCCAGTACAAACACG</td>
<td>CCTGCCAGTACACACG</td>
</tr>
<tr>
<td>Mouse:</td>
<td>0 7 11 10</td>
<td>7 0 4 6</td>
<td>11 4 0 2</td>
<td>10 6 2 0</td>
</tr>
<tr>
<td>Gorilla:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chimpanzee:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Human:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Alignment vs. Distance Matrix

Sequence a gene of length \( m \) in \( n \) species \( \rightarrow n \times m \) alignment matrix.

Reverse transformation not possible due to loss of information.
Distances in Trees

Given a tree $T$ with a positive weight $w(e)$ on each edge, we define the **tree distance** $d_T$ on the set $L$ of leaves by:

$$d_T(i, j) = \text{sum of weights of edges on unique path from } i \text{ to } j.$$ 

In evolutionary biology, weights are sometimes called **branch lengths**.

Distance in Trees: an Example

$$d_T(1,4) = 12 + 13 + 14 + 17 + 13 = 69$$
Distance vs. Tree Distance

- $n \times n$ distance matrix for $n$ species
- Note that $d_T(i,j)$, tree distance between $i$ and $j$, not necessarily equal to $D_{ij}$ as given by distance matrix.

Rat: ACGTGACGCCCAACGCT
Mouse: ACGTGACGCTACAAAAACGT
Gorilla: CCTGTGACCTAAACACGA
Chimpanzee: CCTGTGACCTAGCACAACGA
Human: CCTGTGACCTAGCACAACGA

Fitting a Distance Matrix

- Given $n$ species, we can compute the $n \times n$ distance matrix $D_{ij}$
- Evolution of these species is described by a tree that we don’t know.
- We need an algorithm to construct a tree that best fits the distance matrix $D_{ij}$

Find a tree $T$ such that:

\[
D_{ij} = d_T(i,j)
\]

Distance between species (known)
Distance Based Phylogeny Problem

Goal: Reconstruct an evolutionary tree from a distance matrix

Input: \( n \times n \) distance matrix \( D_{ij} \)

Output: weighted tree \( T \) with \( n \) leaves fitting \( D \)

Unknown topology of tree makes evolutionary tree reconstruction **hard**!

# unrooted binary trees \( n \) leaves: \( T(n) = \frac{(2n-3)!}{((n-2)! \times 2^{n-2})} \)

\( n = 24: T(n) = 5.74 \times 10^{26} \)

If \( D \) is **additive**, this problem has a solution and there is a simple algorithm to solve it.

Distance-based vs. character-based

**Key difference:**

Distance-based methods do not reconstruct ancestral states.

```
   A
   / \1
  /   \0
 A  B  C  D
 A  0   1   2   2
 B  1   0   1   1
 C  2   1   0   0
 D  2   1   0   0
```

Note that C and D are identical.
Reconstructing a 3 Leaved Tree

- Tree reconstruction for a 3x3 matrix is straightforward
- We have 3 leaves $i$, $j$, $k$ and a center vertex $c$

![Diagram of a 3-leaved tree with distances and equations]

Observe:

\[
\begin{align*}
    d_{ic} + d_{jc} &= D_{ij} \\
    d_{ic} + d_{kc} &= D_{ik} \\
    d_{jc} + d_{kc} &= D_{jk}
\end{align*}
\]

Reconstructing a 3 Leaved Tree (cont’d)

\[
\begin{align*}
    d_{ic} + d_{jc} &= D_{ij} \\
    + d_{ic} + d_{kc} &= D_{jk} \\
    2d_{ic} + d_{jc} + d_{kc} &= D_{ij} + D_{ik} \\
    2d_{ic} + D_{jk} &= D_{ij} + D_{ik} \\
    d_{ic} &= (D_{ij} + D_{ik} - D_{jk})/2
\end{align*}
\]

Similarly,

\[
\begin{align*}
    d_{jc} &= (D_{ij} + D_{jk} - D_{ik})/2 \\
    d_{kc} &= (D_{ki} + D_{kj} - D_{ij})/2
\end{align*}
\]
Trees with > 3 Leaves

• A binary tree with $n$ leaves has $2n-3$ edges

• Fitting a given tree to a distance matrix $D$ requires solving a system with $n(n-1)/2$ equations and $2n-3$ variables

• Solution not always possible for $n > 3$.

Additive Distance Matrices

Matrix $D$ is ADDITIVE if there exists a tree $T$ with $d_{ij}(T) = D_{ij}$

Matrix $D$ is NON-ADDITIVE otherwise

\[
\begin{array}{cccc}
A & B & C & D \\
A & 0 & 2 & 4 & 4 \\
B & 2 & 0 & 4 & 4 \\
C & 4 & 4 & 0 & 2 \\
D & 4 & 4 & 2 & 0 \\
\end{array}
\]

\[
\begin{array}{cccc}
A & B & C & D \\
A & 0 & 2 & 2 & 2 \\
B & 2 & 0 & 3 & 2 \\
C & 2 & 3 & 0 & 2 \\
D & 2 & 2 & 2 & 0 \\
\end{array}
\]
Additive Distance Phylogeny

**Small Additive Distance Phylogeny**: Given phylogenetic tree $T$ and distance matrix $D$, determine branch lengths such that $d_{T}(i,j) = D_{ij}$.

**Large Additive Distance Phylogeny**: Given distance matrix $D$, find $T$ and branch lengths such that $d_{T}(i,j) = D_{ij}$.

Both of these problems can be solved efficiently.

Reconstructing Additive Distances

Given $T$

<table>
<thead>
<tr>
<th></th>
<th>v</th>
<th>w</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>0</td>
<td>10</td>
<td>17</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>w</td>
<td>0</td>
<td>15</td>
<td>14</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>0</td>
<td>9</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>0</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If we know $T$ and $D$, but do not know the length of each edge, we can reconstruct those lengths.
Reconstructing Additive Distances Given T

\[
\begin{array}{cccccc}
 & v & w & x & y & z \\
v & 0 & 10 & 17 & 16 & 16 \\
w & 0 & 15 & 14 & 14 \\
x & 0 & 9 & 14 \\
y & 0 & 14 \\
z & 0 \\
\end{array}
\]

Find neighbors v, w (common parent)

\[
\begin{align*}
\text{d}_{ax} &= \frac{1}{2} (d_{ax} + d_{wx} - d_{vw}) \\
\text{d}_{ay} &= \frac{1}{2} (d_{ay} + d_{wy} - d_{vw}) \\
\text{d}_{az} &= \frac{1}{2} (d_{az} + d_{wz} - d_{vw})
\end{align*}
\]
Reconstructing Additive Distances
Given T

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>11</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>x</td>
<td>0</td>
<td>9</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>0</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Neighbors x, y** (common parent)

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

d(a, c) = 3
\[ d(b, c) = d(a, b) - d(a, c) = 3 \]
\[ d(c, z) = d(a, z) - d(a, c) = 7 \]
\[ d(b, x) = d(a, x) - d(a, b) = 5 \]
\[ d(b, y) = d(a, y) - d(a, b) = 4 \]
\[ d(a, w) = d(z, w) - d(a, z) = 4 \]
\[ d(a, v) = d(z, v) - d(a, z) = 6 \]
**Correct!!!**

Trees and Neighbors

Previous algorithm relied only on finding neighboring leaves:

1. Find neighboring leaves \(i\) and \(j\) with parent \(k\)
2. Remove the rows and columns of \(i\) and \(j\)
3. Add a new row and column corresponding to \(k\), where the distance from \(k\) to any other leaf \(m\) can be computed as: \[ D_{km} = \frac{D_{im} + D_{jm} - D_{ij}}{2} \]

Compress \(i\) and \(j\) into \(k\), iterate algorithm for rest of tree
Finding Neighboring Leaves

To find neighboring leaves we simply select a pair of closest leaves.

\[
\begin{array}{cccc}
  & i & j & k & l \\
i & 0 & 13 & 21 & 22 \\
j & 0 & 12 & 13 & \\
k & 0 & 13 & \\
l & 0 & \\
\end{array}
\]

\[d_{ij} = 13 > d_{jk} = 12\]

Finding a pair of neighboring leaves is a nontrivial problem!

Degenerate Triples

- A degenerate triple is a set of three distinct elements \(1 \leq i, j, k \leq n\) where

\[D_{ij} + D_{jk} = D_{ik}\]

- Element \(j\) in a degenerate triple \(i,j,k\) lies on the evolutionary path from \(i\) to \(k\) (or is attached to this path by an edge of length 0).
Looking for Degenerate Triples

- If distance matrix $D$ has a degenerate triple $i,j,k$ then $j$ can be “removed” from $D$ thus reducing the size of the problem.
- If distance matrix $D$ does not have a degenerate triple $i,j,k$, one can “create” a degenerate triple in $D$ by shortening all hanging edges (in the tree).

Shortening Hanging Edges to Produce Degenerate Triples

- Shorten all “hanging” edges (edges that connect leaves) until a degenerate triple is found.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>4</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
<td>0</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>C</td>
<td>10</td>
<td>8</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>D</td>
<td>9</td>
<td>7</td>
<td>9</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ \delta = 1 \]
Finding Degenerate Triples

• If there is no degenerate triple, all hanging edges are reduced by the same amount $\delta$, so that all pair-wise distances in the matrix are reduced by $2\delta$.

• Eventually this process collapses one of the leaves (when $\delta = \text{length of shortest hanging edge}$), forming a degenerate triple $i,j,k$ and reducing the size of the distance matrix $D$.

• The attachment point for $j$ can be recovered in the reverse transformations by saving $D_{ij}$ for each collapsed leaf.

Reconstructing Trees for Additive Distance Matrices

Trim($D$, $\delta$) for all $1 \leq i \neq j \leq n$

$D_{ij} = D_{ij} - 2\delta$
AdditivePhylogeny Algorithm

AdditivePhylogeny(D)
if D is a 2 x 2 matrix
    T = tree of a single edge of length D_{1,2}
    return T
if D is non-degenerate
    Compute trimming parameter δ
    Trim(D, δ)
    Find a triple i, j, k in D such that D_{ij} + D_{jk} = D_{ik}
    x = D_{ij}
    Remove jth row and jth column from D
    T = AdditivePhylogeny(D)
Traceback

AdditivePhylogeny (cont’d)

Traceback
Add a new vertex v to T at distance x from i to k
Add j back to T by creating an edge (v,j) of length 0
for every leaf l in T
    if distance from l to v in the tree ≠ D_{l,j}
        output “matrix is not additive”
return
Extend all “hanging” edges by length δ
return T

Question: How to compute δ?
Additive Distance

• How to tell if D is additive?
• AdditivePhylogeny provides a way to check if distance matrix $D$ is additive

• An even more efficient additivity check is the “four-point condition”

The Four Point Condition
(Zaretskii 1965, Buneman 1971)

Let $1 \leq i,j,k,l \leq n$ be four distinct leaves in a tree

Compute: 1. $D_{ij} + D_{kl}$, 2. $D_{ik} + D_{jl}$, 3. $D_{il} + D_{jk}$

2 and 3 represent the same number: (length of all edges) + 2 * (length middle edge)

1 represents a smaller number: (length of all edges) – (length middle edge)
The Four Point Condition

**Four point condition:** Every four leaves (quartet) can be labeled as $i,j,k,l$ such that:

$$D_{ij} + D_{kl} \leq D_{ik} + D_{jl} = D_{il} + D_{jk}$$

**Theorem:** An $n \times n$ matrix $D$ is additive if and only if the four point condition holds for every quartet $1 \leq i,j,k,l \leq n$.

**Proof:**

Since $D$ additive, $D = d_T$. Find split such that:

$i, j \in S_1$ and $k, l \in S_2$. Define $\lambda_m$ to be weights in tree below.

$$D_{ik} + D_{jl} = (\lambda_1 + \lambda_3 + \lambda_4) + (\lambda_2 + \lambda_3 + \lambda_5) = D_{il} + D_{jk}$$

$$\geq (\lambda_1 + \lambda_2) + (\lambda_4 + \lambda_5).$$

![Diagram of tree with weights](image)
Non-additive Distances

• What if there is no tree $T$ such that $D_{ij} = d_T(i,j)$.

• Approaches:
  1. Find tree such that minimizes “error”

Least Squares Distance Phylogeny Problem

• If the distance matrix $D$ is NOT additive, then we look for a tree $T$ that approximates $D$ the best:

$$\text{Squared Error} : \sum_{i,j} (d_{ij}(T) - D_{ij})^2$$

• Squared Error is a measure of the quality of the fit between distance matrix and the tree: we want to minimize it.

• Least Squares Distance Phylogeny Problem:
  – Find approximation tree $T$ with minimum squared error for a non-additive matrix $D$.
  – (NP-hard)
Tree construction as clustering

Pair Group Methods

Iteratively combine closest leaves/groups into larger groups.

\[ C \leftarrow \{ \{1\}, \ldots, \{n\} \} \]

While \(|C| > 2\) do

[Find closest clusters.]

\[ d(C_i, C_j) = \min d(C_i, C_j). \]

\[ C_k \leftarrow C_i \cup C_j \]

[Replace \(C_i\) and \(C_j\) by \(C_k\).]

\[ C \leftarrow (C \setminus C_i \setminus C_j) \cup C_k. \]
Pair Group Methods

What is $d$?

How to define branch lengths?

$C \leftarrow \{ \{1\}, \ldots, \{n\} \}$

While $|C| > 2$ do

[Find closest clusters.]

$d(C_i, C_j) = \min d(C_i, C_j)$.

$C_k \leftarrow C_i \cup C_j$

[Replace $C_i$ and $C_j$ by $C_k$.]

$C \leftarrow (C \setminus C_i \setminus C_j) \cup C_k$.

UPGMA
Unweighted Pair Group Method with Averages

• Distance between clusters defined as average pairwise distance

• Assigns height to every vertex in the tree, effectively dating every vertex
UPGMA

Unweighted Pair Group Method with Averages

Distance between clusters defined as *average pairwise distance*

Given two disjoint clusters $C_i, C_j$ of sequences,

$$d_{ij} = \frac{1}{|C_i| \times |C_j|} \sum_{p \in C_i, q \in C_j} d_{pq}$$

UPGMA

Assigns *height* to every vertex in the tree, effectively dating every vertex

Add a vertex connecting $C_i, C_j$ *and* place it at height $d_{ij}/2$
UPGMA Algorithm

**Initialization:**
Assign each \( x_i \) to its own cluster \( C_i \)
Define one leaf per sequence, each at height 0

**Iteration:**
Find two clusters \( C_i \) and \( C_j \) such that \( d_{ij} \) is min
Let \( C_k = C_i \cup C_j \)
Add a vertex connecting \( C_i \) and \( C_j \) and place it at height \( d_{ij}/2 \)
Delete \( C_i \) and \( C_j \)

**Termination:**
When a single cluster remains

Trees from UPGMA

UPGMA produces an **ultrametric** tree; distance from the root to any leaf is the same

**The Molecular Clock:**
The evolutionary distance between species \( x \) and \( y \) is twice the Earth time to reach the nearest common ancestor
That is, the molecular clock has constant rate in all species
UPGMA’s Weakness: Example

Correct tree

UPGMA

Ultrametrics

$D_{ij}$ is an ultrametric provided for all species $i, j, k$ (distinct leaves of tree) two of the distances $D_{ij}$, $D_{jk}$ and $D_{ik}$ are equal and $\geq$ the third.

Ex. $d(i,k) = d(j,k) \geq d(i,j)$

Proposition: If $d$ is ultrametric, then $d$ is additive.
Ultrametrics

Both additive distance phylogeny and perfect phylogeny can be reduced to the ultrametric phylogeny problem.

Let $v = \text{row of } D \text{ containing largest entry } m_v$.
Define $D_{ij}' = m_v + \frac{(D_{ij} - D_{vi} - D_{vj})}{2}$

\[ = m_v - \lambda_3 \]

**Theorem:** $D$ is additive if and only if $D'$ is ultrametric.
(See Gusfield, Ch. 17)