Lecture: Bioinformatics II
Phylogenetics

ENS Sacley, 2020

Some slides graciously provided by Daniel Huson & Celine Scornavacca
Phylogenetic Trees - Motivation

I think

[Diagram of a phylogenetic tree with labeled nodes and branches]
Motivation
- study relation between species
- evolution of characteristics
- co-evolution (host-parasite)
- geological migration
- genetic development of viruses/diseases

Phylogenetic Trees - Motivation
Motivation
- study relation between species
- evolution of characteristics
- co-evolution (host-parasite)
- geological migration
- genetic development of viruses/diseases

Evolution
genetic material changes over time
⇒ new species “Branch off”
⇒ “tree of life”
Phylogenetic Trees - Motivation
Phylogenetic Trees - Motivation

Ou et al. (1992), Page et Holmes (1998)
Rooted Phylogenetic Trees

evolution of species over time,
leaves extant,
hypothetical ancestors
possibly branch lengths (time)

Notation

taxon, cluster, triplet

Exercise: use xy|z

Exercise

Time
Rooted Phylogenetic Trees

evolution of species over time,
leaves extant,
hypothetical ancestors
possibly branch lengths (time)

Notation

taxon, cluster, triplet
Rooted Phylogenetic Trees

- Evolution of species over time,
- Leaves extant,
- Hypothetical ancestors
- Possibly branch lengths (time)

Notation

- Taxon, cluster, triplet

Exercise: Use xy|z

\[ LCA(xz) = LCA(yz) > LCA(xy) \] and
\[ |LCA(xy), LCA(xz), LCA(yz)| < 3 \] to prove ab|c + bc|d

\[ \Rightarrow \text{ac|d} \]
Rooted Phylogenetic Trees

evolution of species over time, leaves extant, hypothetical ancestors possibly branch lengths (time)

Notation

taxon, cluster, triplet
Rooted Phylogenetic Trees

evolution of species over time, leaves extant, hypothetical ancestors possibly branch lengths (time)

Notation
taxon, cluster, triplet

"Polytomies"
history not clear \(\leadsto\) "soft" known "fan out" \(\leadsto\) "hard"

Exercise: use \(xy|z\) \(\leftrightarrow\) \(LCA(xz) = LCA(yz) > LCA(xy)\) and \(|LCA(xy),LCA(xz),LCA(yz)|<3\) to prove \(ab|c + bc|d\) \(\rightarrow\) \(ac|d\)
Rooted Phylogenetic Trees

- Evolution of species over time
- Leaves extant
- Hypothetical ancestors
- Possibly branch lengths (time)

Notation

- Taxon, cluster, triplet

"Polytomies"

- History not clear \( \rightsquigarrow \) "soft"
- Known "fan out" \( \rightsquigarrow \) "hard"

Exercise: use \( xy|z \leftrightarrow LCA(xz) = LCA(yz) > LCA(xy) \) and \( |LCA(xy),LCA(xz),LCA(yz)|<3 \) to prove \( ab|c + bc|d \rightarrow ac|d \)
Unrooted Phylogenetic Trees

similarity between genomes,
leaves extant,
internal vertices have no meaning
possibly branch lengths (amount of change)

Notation

taxon, split, quartet
Unrooted Phylogenetic Trees

similarity between genomes,
leaves extant,
internal vertices have no meaning
possibly branch lengths (amount of change)

Notation

taxon, split, quartet
Unrooted Phylogenetic Trees

similarity between genomes,
leaves extant,
internal vertices have **no meaning**
possibly branch lengths (amount of change)

**Notation**
taxon, split, quartet
Unrooted Phylogenetic Trees

similarity between genomes,
leaves extant,
internal vertices have no meaning
possibly branch lengths (amount of change)

Notation

taxon, split, quartet
Unrooted Phylogenetic Trees

similarity between genomes, leaves extant, internal vertices have no meaning possibly branch lengths (amount of change)

Notation

taxon, split, quartet
Unrooted Phylogenetic Trees

similarity between genomes,
leaves extant,
internal vertices have no meaning
possibly branch lengths (amount of change)

Notation

taxon, split, quartet

Note: unrooted trees can be rooted at any edge
Reconstructing Phylogenetic Trees

Group Species By...
- morphology
- behavior
- geography

Diptera = two wings
Reconstructing Phylogenetic Trees

Group Species By...
- morphology
- behavior
- geography
- distance of sequences
- "genetic distance"
- etc.

Diptera = two wings
Reconstructing Phylogenetic Trees

Vertebrata
has backbone
Reconstructing Phylogenetic Trees

Vertebrata
has backbone

Tetrapoda
has 4 legs

Keep in Mind
Automatic reconstruction should be
- fast (deal with tons of species & genes)
- consistent (optimal data ⇝ evolution correctly reflected)
- non-arbitrary
Reconstructing Phylogenetic Trees

Vertebrata
has backbone

Tetrapoda
has 4 legs

Mammalia
breast feeding

Keep in Mind
Automatic reconstruction should be:
- fast (deal with tons of species & genes)
- consistent (optimal data $\rightarrow$ evolution correctly reflected)
- non-arbitrary
Reconstructing Phylogenetic Trees

Vertebrata
- has backbone

Tetrapoda
- has 4 legs

Salientia
- can leap

Mammalia
- breast feeding

Keep in Mind
- Automatic reconstruction should be
  - fast (deal with tons of species & genes)
  - consistent (optimal data $\Rightarrow$ evolution correctly reflected)
  - non-arbitrary
Reconstructing Phylogenetic Trees

Vertebrata
- has backbone

Tetrapoda
- has 4 legs

Salientia
- can leap

Mammalia
- breast feeding

Elephantidae
- elephants

Asian Elephant
- Elephas maximus

African Elephant
- Loxodonta africana

Keep in Mind
- Automatic reconstruction should be:
  - fast (deal with tons of species & genes)
  - consistent (optimal data ⇝ evolution correctly reflected)
  - non-arbitrary
Reconstructing Phylogenetic Trees

Vertebrata
has backbone

Tetrapoda
has 4 legs

Salientia
can leap

Mammalia
Breast feeding

Elephantidae
elephants

Loxodonta africana
African Elephant

Elephas maximus
Asian Elephant
Reconstructing Phylogenetic Trees

Vertebrata
  has backbone

Tetrapoda
  has 4 legs

Salientia
  can leap

Mammalia
  Breast feeding

Elephantidae
  elephants

Loxodonta africana
  African Elephant

Elephas maximus
  Asian Elephant

**Keep in Mind**

Automatic reconstruction should be
- fast (deal with tons of species & genes)
- consistent (optimal data \(\rightarrow\) evolution correctly reflected)
- non-arbitrary
Distance-Based Reconstruction

Let's Start Simple

1. Assume distances are perfect ("additive")
2. Assume ultrametric ("molecular clock")

Idea
1. Merge closest taxa
2. Update distances
3. Recurse
Let's Start Simple

1. assume distances are perfect ("additive")
2. assume ultrametric ("molecular clock")

Idea
1. merge closest taxa
2. update distances
3. recurse
Distance-Based Reconstruction

Let's Start Simple

1. assume distances are perfect ("additive")
2. assume ultrametric ("molecular clock")
Distance-Based Reconstruction

Let's Start Simple

1. assume distances are perfect ("additive")
2. assume ultrametric ("molecular clock")

Idea

1. merge closest taxa
2. update distances
3. recurse
Distance-Based Reconstruction

Let's Start Simple

1. assume distances are perfect ("additive")
2. assume ultrametric ("molecular clock")

Idea

1. merge closest taxa
2. update distances
3. recurse

⇒ ✓ ⇐ ?

Let's Start Simple

1. assume distances are perfect ("additive")
2. assume ultrametric ("molecular clock")

Idea

1. merge closest taxa
2. update distances
3. recurse

⇒ ✓ ⇐ ?
Distance-Based Reconstruction

Let's Start Simple

1. assume distances are perfect ("additive")
2. assume ultrametric ("molecular clock")

Idea

1. merge closest taxa
2. update distances
3. recurse
Distance-Based Reconstruction

**Let’s Start Simple**

1. assume distances are perfect (“additive”)
2. assume ultrametric (“molecular clock”)

**Idea**

1. merge closest taxa
2. update distances
3. recurse
Distance-Based Reconstruction

Let's Start Simple

1. assume distances are perfect ("additive")
2. assume ultrametric ("molecular clock")

Idea

1. merge closest taxa
2. update distances
3. recurse
Distance-Based Reconstruction

Let's Start Simple

1. assume distances are perfect ("additive")
2. assume ultrametric ("molecular clock")

Idea

1. merge closest taxa
2. update distances
3. recurse
Distance-Based Reconstruction

Let's Start Simple

1. assume distances are perfect ("additive")
2. assume ultrametric ("molecular clock")

Idea

1. merge closest taxa
2. update distances
3. recurse
Distance-Based Reconstruction

Let's Start Simple

1. assume distances are perfect ("additive")
2. assume ultrametric ("molecular clock")

Idea

1. merge closest taxa
2. update distances
3. recurse
Distance-Based Reconstruction

Let's Start Simple

1. assume distances are perfect ("additive")
2. assume ultrametric ("molecular clock")

Idea

1. merge closest taxa
2. update distances
3. recurse
Distance-Based Reconstruction

Let's Start Simple

1. assume distances are perfect ("additive")
2. assume ultrametric ("molecular clock")

Idea

1. merge closest taxa
2. update distances
3. recurse
Distance-Based Reconstruction

Let's Start Simple

1. assume distances are perfect ("additive")
2. assume ultrametric ("molecular clock")

Idea

1. merge closest taxa
2. update distances
3. recurse
Distance-Based Reconstruction

Let’s Start Simple

1. assume distances are perfect ("additive")
2. assume ultrametric ("molecular clock")

Idea

1. merge closest taxa
2. update distances
3. recurse
Distance-Based Reconstruction

Let’s Start Simple

1. assume distances are perfect (“additive”)
2. assume ultrametric (“molecular clock”)

Idea

1. merge closest taxa
2. update distances
3. recurse
Unweighted Pair Group Method w/ Avg.

Idea

1. merge closest taxa
2. update distances
3. recurse
Unweighted Pair Group Method w/ Avg.

Idea

1. merge closest taxa
2. update distances
3. recurse

new dist: weighted avg.

\[ dx_{X \cup Y, Z} = \frac{|X|d_{X, Z} + |Y|d_{Y, Z}}{|X| + |Y|} \]
Unweighted Pair Group Method w/ Avg.

new dist: weighted avg.

\[ d_{X \cup Y, Z} = \frac{|X| d_{X, Z} + |Y| d_{Y, Z}}{|X| + |Y|} \]

Idea

1. merge closest taxa
2. update distances
3. recurse
Unweighted Pair Group Method w/ Avg.

new dist: weighted avg.
\[ d_{X \cup Y, Z} = \frac{|X| d_{X, Z} + |Y| d_{Y, Z}}{|X| + |Y|} \]

**Idea**

1. merge closest taxa
2. update distances
3. recurse
Unweighted Pair Group Method w/ Avg.

**New dist: weighted avg.**

\[ d_{XUY,Z} = \frac{|X|d_{X,Z} + |Y|d_{Y,Z}}{|X| + |Y|} \]

**Idea**

1. merge closest taxa
2. update distances
3. recurse
Unweighted Pair Group Method w/ Avg.

new dist: weighted avg.

\[ d_{X \cup Y, Z} = \frac{|X|d_{X, Z} + |Y|d_{Y, Z}}{|X| + |Y|} \]

Idea

1. merge closest taxa
2. update distances
3. recurse
**Unweighted Pair Group Method w/ Avg.**

```
11
8 7
6 11 8
9 4 5 9
```

**new dist: weighted avg.**

\[
d_{X \cup Y, Z} = \frac{|X|d_{X, Z} + |Y|d_{Y, Z}}{|X| + |Y|}
\]

**Idea**

1. merge closest taxa
2. update distances
3. recurse
Unweighted Pair Group Method w/ Avg.

**Idea**

1. merge closest taxa
2. update distances
3. recurse

**new dist: weighted avg.**

\[ d_{X\cup Y, Z} = \frac{|X|d_{X, Z} + |Y|d_{Y, Z}}{|X| + |Y|} \]
Unweighted Pair Group Method w/ Avg.

Idea
1. merge closest taxa
2. update distances
3. recurse

new dist: weighted avg.
\[ d_{X \cup Y, Z} = \frac{|X| d_{X, Z} + |Y| d_{Y, Z}}{|X| + |Y|} \]
Unweighted Pair Group Method w/ Avg.

new dist: weighted avg.

\[ d_{X \cup Y, Z} = \frac{|X|d_{X,Z} + |Y|d_{Y,Z}}{|X| + |Y|} \]

Idea

1. merge closest taxa
2. update distances
3. recurse
Unweighted Pair Group Method w/ Avg.

**new dist: weighted avg.**

\[ d_{X \cup Y, Z} = \frac{|X| d_{X, Z} + |Y| d_{Y, Z}}{|X| + |Y|} \]

**Idea**

1. merge closest taxa
2. update distances
3. recurse
Unweighted Pair Group Method with Avg.

new dist: weighted avg.

\[ d_{X \cup Y, Z} = \frac{|X|d_{X,Z} + |Y|d_{Y,Z}}{|X| + |Y|} \]

Idea

1. merge closest taxa
2. update distances
3. recurse
Unweighted Pair Group Method w/ Avg.

new dist: weighted avg.
\[ d_{X \cup Y, Z} = \frac{|X|d_{X, Z} + |Y|d_{Y, Z}}{|X| + |Y|} \]

Idea
1. merge closest taxa
2. update distances
3. recurse
Unweighted Pair Group Method w/ Avg.

**Idea**
1. merge closest taxa
2. update distances
3. recurse

**new dist: weighted avg.**

\[ d_{X \cup Y, Z} = \frac{|X|d_{X,Z} + |Y|d_{Y,Z}}{|X| + |Y|} \]
Unweighted Pair Group Method with Avg.

**Idea**
1. merge closest taxa
2. update distances
3. recurse

new dist: weighted avg.

\[ d_{X \cup Y, Z} = \frac{|X|d_{X,Z} + |Y|d_{Y,Z}}{|X| + |Y|} \]
Unweighted Pair Group Method w/ Avg.

### Idea
1. merge closest taxa
2. update distances
3. recurse

### new dist: weighted avg.

\[
d_{X \cup Y, Z} = \frac{|X|d_{X, Z} + |Y|d_{Y, Z}}{|X| + |Y|}
\]
**Unweighted Pair Group Method w/ Avg.**

New dist: weighted avg.

\[ d_{X \cup Y, Z} = \frac{|X|d_{X, Z} + |Y|d_{Y, Z}}{|X| + |Y|} \]

**Idea**

1. merge closest taxa
2. update distances
3. recurse
Non-Additive Phylogeny

Let’s Start Simple

1. assume distances are perfect ("additive")
2. assume ultrametric ("molecular clock")
Non-Additive Phylogeny

Let’s Start Simple

1. assume distances are perfect (“additive”)
2. assume ultrametric (“molecular clock”)

Note: no molecular clock $\Rightarrow$ cannot reconstruct rooted tree
$\Rightarrow$ unrooted tree
Non-Additive Phylogeny

Let’s Start Simple

1. assume distances are perfect (“additive”)
2. assume ultrametric (“molecular clock”)

Note: no molecular clock $\not\Rightarrow$ cannot reconstruct rooted tree
$\not\Rightarrow$ unrooted tree

Problem 1: correct pairs may not be closest
Non-Additive Phylogeny

Let's Start Simple

1. assume distances are perfect (“additive”)
2. assume ultrametric (“molecular clock”)

Note: no molecular clock → cannot reconstruct rooted tree → unrooted tree

Problem 1: correct pairs may not be closest
Non-Additive Phylogeny

Let’s Start Simple

1. assume distances are perfect (“additive”)
2. assume ultrametric (“molecular clock”)

**Note:** no molecular clock $\Rightarrow$ cannot reconstruct rooted tree
$\Rightarrow$ unrooted tree

**Problem 1:** correct pairs may not be closest
**Problem 2:** how to update distances without molecular clock?
Non-Additive Phylogeny

Let’s Start Simple

1. assume distances are perfect (”additive”)
2. assume ultrametric (”molecular clock”)

**Note**: no molecular clock $\Rightarrow$ cannot reconstruct rooted tree $\Rightarrow$ unrooted tree

**Problem 1**: correct pairs may not be closest

**Problem 2**: how to update distances without molecular clock?

\[
\begin{align*}
X & \quad \text{(root)} \\
A & \quad \text{B} \\
& \quad \text{C} \\
\end{align*}
\]
Non-Additive Phylogeny

Let’s Start Simple

1. assume distances are perfect (“additive”)
2. assume ultrametric (“molecular clock”)

Note: no molecular clock $\Rightarrow$ cannot reconstruct rooted tree
$\Rightarrow$ unrooted tree

Problem 1: correct pairs may not be closest
Problem 2: how to update distances without molecular clock?

\[ d_{X,C} = \frac{1}{2}(d_{A,C} + d_{B,C} - d_{A,B}) \]
Non-Additive Phylogeny

Let’s Start Simple

1. assume distances are perfect (“additive”)
2. assume ultrametric (“molecular clock”)

Note: no molecular clock $\Rightarrow$ cannot reconstruct rooted tree
$\Rightarrow$ unrooted tree

Problem 1: correct pairs may not be closest

Problem 2: how to update distances without molecular clock?
Non-Additive Phylogeny

Let’s Start Simple

1. assume distances are perfect (“additive”)
2. assume ultrametric (“molecular clock”)

Note: no molecular clock $\leadsto$ cannot reconstruct rooted tree $\leadsto$ unrooted tree

Problem 1: correct pairs may not be closest
Problem 2: how to update distances without molecular clock?

$$d_{AC} + d_{BC} = 2d_{XC} + d_{AB}$$
Non-Additive Phylogeny

Let's Start Simple

1. assume distances are perfect ("additive")

2. assume ultrametric ("molecular clock")

Note: no molecular clock \(\rightarrow\) cannot reconstruct rooted tree
\(\rightarrow\) unrooted tree

Problem 1: correct pairs may not be closest

Problem 2: how to update distances without molecular clock?

\[
\begin{align*}
\text{Problem 2:} & \quad d_{AC} + d_{BC} = 2d_{XC} + d_{AB} \\
& \quad d_{XC} = \frac{1}{2}(d_{AC} + d_{BC} - d_{AB})
\end{align*}
\]
Neighbor Joining

\[ Q_{X,Y} = \sum_{Z \in \{X,Y\}} (d_{X,Z} + d_{Y,Z} - d_{X,Y}) + 2d_{X,Y} \]

- find max in \( Q \)
- join corresponding indices
- update distances & recurse

**Theorem**

\[ Q_{X,Y} \text{ max} \iff X \& Y \text{ "neighbors" in any tree } T \]

\[ d_{X \cup Y,Z} = \frac{1}{2} (d_{X,Z} + d_{Y,Z} - d_{X,Y}) \]

\[ d_{X \cup Y,Z} = \frac{1}{n-2} \sum_{Z \in \{X,Y\}} (d_{X,Z} + d_{Y,Z} - d_{X,Y}) \]
Neighbor Joining

\[ Q_{X,Y} = \sum_{Z} (d_{X,Z} + d_{Y,Z} - d_{X,Y}) + 2d_{X,Y} \]

- find max in \( Q \)
- join corresponding indices
- update distances & recurse

**Theorem**

\[ Q_{X,Y} \text{ max} \iff X \& Y \text{ "neighbors" in any tree } T \text{ yielding } Q \text{ branch lengths } d_{X \cup Y,Z} = \frac{1}{2}(d_{X,Z} + d_{Y,Z} - d_{X,Y}) \]
**Neighbor Joining**

**Neighbor Joining (unrooted)**

- Build new matrix:
  \[ Q_{X,Y} = \sum_Z (d_{X,Z} + d_{Y,Z} - d_{X,Y}) + 2d_{X,Y} \]
- Find max in \( Q \)
- Join corresponding indices
- Update distances \& recurse

**Theorem**

\[ Q_{X,Y} \text{ max} \iff X \neq Y \text{ "neighbors" in any tree } T \text{ yielding } Q \]
Neighbor Joining

**Neighbor Joining (unrooted)**

- Build new matrix: 
  \[ Q_{X,Y} = \sum_Z (d_{X,Z} + d_{Y,Z} - d_{X,Y}) + 2d_{X,Y} \]
- Find max in \( Q \)
- Join corresponding indices
- Update distances & recurse

---

**Branch lengths**

\[ d_{X\cup Y,X} = \frac{1}{n-2} \sum_{Z \notin \{X,Y\}} (d_{X,Z} + d_{X,Y} - d_{Y,Z}) \]

**Update distances**

\[ d_{X\cup Y,Z} = \frac{1}{2} (d_{X,Z} + d_{Y,Z} - d_{X,Y}) \]
Neighbor Joining

**Neighbor Joining (unrooted)**

- Build new matrix:
  \[ Q_{X,Y} = \sum_{Z} (d_{X,Z} + d_{Y,Z} - d_{X,Y}) + 2d_{X,Y} \]
- Find max in \( Q \)
- Join corresponding indices
- Update distances \& recurse

**Branch lengths**

\[ d_{X\cup Y, X} = \frac{1}{n-2} \sum_{Z \notin \{X,Y\}} (d_{X,Z} + d_{X,Y} - d_{Y,Z}) \]

**Update distances**

\[ d_{X\cup Y, Z} = \frac{1}{2} (d_{X,Z} + d_{Y,Z} - d_{X,Y}) \]
**Neighbor Joining**

**Neighbor Joining (unrooted)**
- Build new matrix:
  \[ Q_{X,Y} = \sum_Z (d_{X,Z} + d_{Y,Z} - d_{X,Y}) + 2d_{X,Y} \]
- Find max in \( Q \)
- Join corresponding indices
- Update distances & recurse

**Theorem**
\[ Q_{X,Y} \text{ max} \iff X \& Y \text{ "neighbors" in any tree } T \text{ yielding branch lengths} \]

**Branch lengths**
\[ d_{X \cup Y, X} = \frac{1}{n-2} \sum_{Z \notin \{X, Y\}} (d_{X,Z} + d_{X,Y} - d_{Y,Z}) \]

**Update distances**
\[ d_{X \cup Y, Z} = \frac{1}{2} (d_{X,Z} + d_{Y,Z} - d_{X,Y}) \]
Neighbor Joining

- Build new matrix:
  \[ Q_{X,Y} = \sum_Z (d_{X,Z} + d_{Y,Z} - d_{X,Y}) + 2d_{X,Y} \]
- find max in \( Q \)
- join corresponding indices
- update distances & recurse

Branch lengths

\[ d_{X \cup Y,x} = \frac{1}{n-2} \sum_{Z \notin \{X,Y\}} (d_{X,Z} + d_{X,Y} - d_{Y,Z}) \]

update distances

\[ d_{X \cup Y,z} = \frac{1}{2} (d_{X,Z} + d_{Y,Z} - d_{X,Y}) \]
Neighbor Joining

**Neighbor Joining (unrooted)**
- Build new matrix:
  \[ Q_{X,Y} = \sum_Z (d_{X,Z} + d_{Y,Z} - d_{X,Y}) + 2d_{X,Y} \]
- find max in \( Q \)
- join corresponding indices
- update distances & recurse

**Branch lengths**
\[ d_{X \cup Y,X} = \frac{1}{n-2} \sum_{Z \notin \{X,Y\}} (d_{X,Z} + d_{X,Y} - d_{Y,Z}) \]

**Update distances**
\[ d_{X \cup Y,Z} = \frac{1}{2} (d_{X,Z} + d_{Y,Z} - d_{X,Y}) \]
Neighbor Joining

**Neighbor Joining (unrooted)**

- Build new matrix:
  \[ Q_{x,y} = \sum_{z} (d_{x,z} + d_{y,z} - d_{x,y}) + 2d_{x,y} \]
- Find max in \( Q \)
- Join corresponding indices
- Update distances \& recurse

**Branch lengths**

\[ d_{X \cup Y, X} = \frac{1}{n-2} \sum_{Z \notin \{X,Y\}} (d_{X,Z} + d_{X,Y} - d_{Y,Z}) \]

**Update distances**

\[ d_{X \cup Y, Z} = \frac{1}{2} (d_{X,Z} + d_{Y,Z} - d_{X,Y}) \]
Neighbor Joining

Neighbor Joining (unrooted)

- Build new matrix:
  \[ Q_{X,Y} = \sum_Z (d_{X,Z} + d_{Y,Z} - d_{X,Y}) + 2d_{X,Y} \]
- Find max in \( Q \)
- Join corresponding indices
- Update distances & recurse

Branch lengths

\[ d_{X \cup Y,Z} = \frac{1}{n-2} \sum_{Z \notin \{X,Y\}} (d_{X,Z} + d_{X,Y} - d_{Y,Z}) \]

Update distances

\[ d_{X \cup Y,Z} = \frac{1}{2} (d_{X,Z} + d_{Y,Z} - d_{X,Y}) \]
Neighbor Joining

Neighbor Joining (unrooted)

- Build new matrix:
  \[ Q_{X,Y} = \sum_Z (d_{X,Z} + d_{Y,Z} - d_{X,Y}) + 2d_{X,Y} \]
- Find max in \( Q \)
- Join corresponding indices
- Update distances \& recurse

Branch lengths

\[ d_{X\cup Y,X} = \frac{1}{n-2} \sum_{Z \notin \{X,Y\}} (d_{X,Z} + d_{X,Y} - d_{Y,Z}) \]

Update distances

\[ d_{X\cup Y,Z} = \frac{1}{2} (d_{X,Z} + d_{Y,Z} - d_{X,Y}) \]
Parsimony Reconstructing

<table>
<thead>
<tr>
<th>fur</th>
<th>AUS</th>
<th>pouch</th>
<th>land</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Small Parsimony
Input: character state matrix M, rooted tree T
Task: assign characters to internal nodes minimizing total cost
\[ \Rightarrow \text{O(nm) time} \]

Large Parsimony
Input: character state matrix M
Task: find tree T & assign characters to internal nodes minimizing total cost
\[ \Rightarrow \text{NP-hard} \]

Note character-state matrix could be multiple-alignment
\[ \Rightarrow \text{good alignment crucial (columns treated individually)} \]
### Parsimony Reconstructing

<table>
<thead>
<tr>
<th>Character</th>
<th>fur</th>
<th>AUS</th>
<th>pouch</th>
<th>land</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monkey</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Ape</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Dolphin</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Hedgehog</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

#### Small Parsimony
- **Input:** character state matrix $M$, rooted tree $T$
- **Task:** assign characters to internal nodes minimizing total cost
- **Time Complexity:** $O(nm)$ time

#### Large Parsimony
- **Input:** character state matrix $M$
- **Task:** find tree $T$ and assign characters to internal nodes minimizing total cost
- **Complexity:** NP-hard

#### Note
- Character-state matrix could be a multiple-alignment
- Good alignment crucial (columns treated individually)

#### Exercise
- Calculate the cost of the tree using the Hamming distance:
  - Sum the "distance" of endpoints of each edge
  - Cost: 6

---

11 / 38
Parsimony Reconstructing

<table>
<thead>
<tr>
<th>fur</th>
<th>AUS</th>
<th>pouch</th>
<th>land</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Small Parsimony
Input: character state matrix M, rooted tree T
Task: assign characters to internal nodes minimizing total cost
\[ O(nm) \text{ time} \] [Fitch'71]

Large Parsimony
Input: character state matrix M
Task: find tree T & assign characters to internal nodes minimizing total cost
\[ \text{NP-hard} \] [Graham & Foulds,'82]

Note: character-state matrix could be multiple-alignment
\[ \text{good alignment crucial (columns treated individually)} \]

Sum "distance" of endpoints of each edge
\[ \text{cost 6 (Hamming)} \]

Exercise
Parsimony Reconstructing

<table>
<thead>
<tr>
<th>fur</th>
<th>AUS</th>
<th>pouch</th>
<th>land</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Small Parsimony
Input: character state matrix M, rooted tree T
Task: assign characters to internal nodes minimizing total cost
\(\Rightarrow O(nm)\) time
[Fitch'71]

Large Parsimony
Input: character state matrix M
Task: find tree T & assign characters to internal nodes minimizing total cost
\(\Rightarrow \text{NP-hard}\)
[Graham & Foulds,'82]

Note: character-state matrix could be multiple-alignment
\(\Rightarrow \) good alignment crucial (columns treated individually)

sum “distance” of endpoints of each edge
Parsimony Reconstructing

<table>
<thead>
<tr>
<th>fur</th>
<th>AUS</th>
<th>pouch</th>
<th>land</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Small Parsimony
Input: character state matrix $M$, rooted tree $T$
Task: assign characters to internal nodes minimizing total cost
$\Rightarrow O(nm)$ time
[Fitch'71]

Large Parsimony
Input: character state matrix $M$
Task: find tree $T$ & assign characters to internal nodes minimizing total cost
$\Rightarrow$ NP-hard
[Graham & Foulds,'82]

Note character-state matrix could be multiple-alignment $\Rightarrow$ good alignment crucial (columns treated individually)

Exercise
Time 11 / 38

sum "distance" of endpoints of each edge $\Rightarrow$ cost 6 (Hamming)
Parsimony Reconstructing

<table>
<thead>
<tr>
<th>fur</th>
<th>AUS</th>
<th>pouch</th>
<th>land</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Small Parsimony

Input: character state matrix $M$, rooted tree $T$

Task: assign characters to internal nodes minimizing total cost

Large Parsimony

Input: character state matrix $M$

Task: find tree $T$ & assign characters to internal nodes minimizing total cost

Note: character-state matrix could be multiple-alignment

⇒ good alignment crucial (columns treated individually)

Sum “distance” of endpoints of each edge

Cost: $6$ (Hamming)

Exercise

Time: 11 / 38
Fitch’s Algorithm

Idea 1: each character $i$ individually

Idea 2: bottom-up dynamic programming

For each node $u$: states $S[u]$ incurring minimum cost in sub-tree $u$ leaf

$S[u] = M[i,u]$

If $S[v]$ and $S[w]$ intersect

$S[u] = S[v] \cap S[w]$

Otherwise

$S[u] = S[v] \cup S[w]$
Fitch’s Algorithm

Idea 1: each character i individually

Idea 2: bottom-up dynamic programming

For each node u, the states $S[u]$ incurring minimum cost in the sub-tree $u$ leaf

- If $u$ has children $v$ and $w$, then $S[u] = S[v] \cap S[w]$ if $S[v]$ and $S[w]$ intersect.

12 / 38
Fitch’s Algorithm

Idea 1: each character i individually
Idea 2: bottom-up dynamic programming
for each node u: states S[u] incurring minimum cost in sub-tree
Fitch's Algorithm

Idea 1: each character $i$ individually

Idea 2: bottom-up dynamic programming

for each node $u$: states $S[u]$ incurring minimum cost in sub-tree $u$ leaf $\leadsto S[u] = M[i,u]$
Fitch’s Algorithm

Idea 1: each character individually
Idea 2: bottom-up dynamic programming
for each node u: states S[u] incurring minimum cost in sub-tree
u leaf \(\leadsto S[u] = M[i,u]\)
u has children v, w \(\leadsto S[u] = \begin{cases} S[v] \cap S[w] & \text{if } S[v] \text{ and } S[w] \text{ intersect} \\ S[v] \cup S[w] & \text{otherwise} \end{cases}\)
Fitch's Algorithm

| Idea 1: each character i individually |
| Idea 2: bottom-up dynamic programming |

for each node u: states $S[u]$ incurring minimum cost in sub-tree

- **u leaf**: $S[u] = M[i,u]$  
- **u has children v,w**: $S[u] = S[v] \cap S[w]$ if $S[v]$ and $S[w]$ intersect  
  $S[v] \cup S[w]$ otherwise

$$S[u] = \begin{cases} 
S[v] \cap S[w] & \text{if } S[v] \text{ and } S[w] \text{ intersect} \\
S[v] \cup S[w] & \text{otherwise}
\end{cases}$$
Fitch’s Algorithm

Idea 1: each character i individually
Idea 2: bottom-up dynamic programming

for each node u: states $S[u]$ incurring minimum cost in sub-tree

- u leaf $\Rightarrow S[u] = M[i,u]$
- u has children v, w $\Rightarrow$

$$S[u] = \begin{cases} S[v] \cap S[w] & \text{if } S[v] \text{ and } S[w] \text{ intersect} \\ S[v] \cup S[w] & \text{otherwise} \end{cases}$$
Fitch’s Algorithm

Idea 1: each character i individually
Idea 2: bottom-up dynamic programming

for each node u: states $S[u]$ incurring minimum cost in sub-tree

- u leaf $\sim S[u] = M[i,u]$
- u has children v,w $\sim$

$$S[u] = \begin{cases} S[v] \cap S[w] & \text{if } S[v] \text{ and } S[w] \text{ intersect} \\ S[v] \cup S[w] & \text{otherwise} \end{cases}$$
Fitch’s Algorithm

Idea 1: each character i individually
Idea 2: bottom-up dynamic programming
for each node u: states \( S[u] \) incurring minimum cost in sub-tree
\( u \) leaf \( \Rightarrow \) \( S[u] = M[i,u] \)
\( u \) has children v,w \( \Rightarrow \)

\[
S[u] = \begin{cases} 
S[v] \cap S[w] & \text{if } S[v] \text{ and } S[w] \text{ intersect} \\
S[v] \cup S[w] & \text{otherwise}
\end{cases}
\]
Parsimony Reconstructing

**Small Parsimony**

**Input:** character state matrix $M$, rooted tree $T$

**Task:** assign characters to internal nodes minimizing total cost

$\sim O(nm)$ time

[Fitch’71]

**Large Parsimony**

**Input:** character state matrix $M$

**Task:** find tree $T$ & assign characters to internal nodes minimizing total cost

$\sim$ NP-hard

[Graham & Foulds,’82]

Note: character-state matrix could be multiple-alignment

$\sim$ good alignment crucial (columns treated individually)

sum “distance” of endpoints of each edge

$\sim$ cost 6 (Hamming)

Exercise

Time
Parsimony Reconstructing

### Small Parsimony

*Input:* character state matrix $M$, rooted tree $T$

*Task:* assign characters to internal nodes minimizing total cost

$\Rightarrow O(nm)$ time

[Fitch'71]

### Large Parsimony

*Input:* character state matrix $M$

*Task:* find tree $T$ and assign characters to internal nodes minimizing total cost

$\Rightarrow$ NP-hard

[Graham & Foulds,'82]

---

**Exercise Time**

<table>
<thead>
<tr>
<th>Legs</th>
<th>Size</th>
<th>Diet</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>15</td>
<td>carni</td>
</tr>
<tr>
<td>2</td>
<td>170</td>
<td>omni</td>
</tr>
<tr>
<td>0</td>
<td>170</td>
<td>carni</td>
</tr>
<tr>
<td>2</td>
<td>80</td>
<td>herbi</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>herbi</td>
</tr>
</tbody>
</table>
Parsimony Reconstructing

**Small Parsimony**
- **Input:** character state matrix $M$, rooted tree $T$
- **Task:** assign characters to internal nodes minimizing total cost

$\Rightarrow O(nm)$ time

[Fitch'71]

**Large Parsimony**
- **Input:** character state matrix $M$
- **Task:** find tree $T$ and assign characters to internal nodes minimizing total cost

Note: character-state matrix could be multiple-alignment $\Rightarrow$ good alignment crucial (columns treated individually)

Sum "distance" of endpoints of each edge $\Rightarrow$ cost 6 (Hamming)

Exercise

Time 13 / 38
Parsimony Reconstructing

Small Parsimony
Input: character state matrix $M$, rooted tree $T$
Task: assign characters to internal nodes minimizing total cost
$\sim O(nm)$ time
[Fitch'71]

Large Parsimony
Input: character state matrix $M$
Task: find tree $T$ and assign characters to internal nodes minimizing total cost
$\sim$ NP-hard
[Graham & Foulds,'82]
Parsimony Reconstructing

Small Parsimony
Input: character state matrix M, rooted tree T
Task: assign characters to internal nodes minimizing total cost
\( \sim O(nm) \) time \[\text{[Fitch'71]}\]

Large Parsimony
Input: character state matrix M
Task: find tree T \& assign characters to internal nodes minimizing total cost
\( \sim \) NP-hard \[\text{[Graham \& Foulds,'82]}\]

Note
character-state matrix could be multiple-alignment
\( \sim \) good alignment crucial (columns treated individually)
Maximum Likelihood Reconstruction

**Idea:** for a given tree $T$ (with branch lengths), maximize the probability that evolution along $T$ produces observed sequences.

Jukes & Cantor Model:
- Each base evolves individually.
- Each base occurs with equal frequency in the genome.
- Constant rate $\mu$ of mutation.
- Each base is equally likely to be the result of mutation.
Maximum Likelihood Reconstruction

Idea: for a given tree $T$ (with branch lengths), maximize the probability that evolution along $T$ produces observed sequences.

$\Rightarrow$ need model of evolution
Maximum Likelihood Reconstruction

**Idea:** for a given tree $T$ (with branch lengths), maximize the probability that evolution along $T$ produces observed sequences.

**Jukes & Cantor Model**

- each base evolves individually
- each base occurs with equal frequency in the genome
- constant rate $\mu$ of mutation
- each base is equally likely to be result of mutation
Maximum Likelihood Reconstruction

Idea: for a given tree $T$ (with branch lengths), maximize the probability that evolution along $T$ produces observed sequences.

Jukes & Cantor Model

- each base evolves individually
- each base occurs with equal frequency in the genome
- constant rate $\mu$ of mutation
- each base is equally likely to be result of mutation

arc uv of length $t$
\[ \sim \# \text{flips poisson distributed (mean } \mu t) \]

$$P(\text{no mutation}) = e^{-\mu t}$$

$$P_{X \rightarrow Y}(t) = \begin{cases} e^{-\mu t} + (1 - e^{-\mu t})\pi_{X \rightarrow X} & \text{if } X = Y \\ (1 - e^{-\mu t})\pi_{X \rightarrow Y} & \text{if } X \neq Y \end{cases}$$
Maximum Likelihood Reconstruction

Idea: for a given tree $T$ (with branch lengths), maximize the probability that evolution along $T$ produces observed sequences.

Jukes & Cantor Model

- each base evolves individually
- each base occurs with equal frequency in the genome
- constant rate $\mu$ of mutation
- each base is equally likely to be result of mutation

arc $uv$ of length $t$

$\sim \$ flips poisson distributed (mean $\mu t$)

$P(\text{no mutation}) = e^{-\mu t}$

$P_{X\rightarrow Y}(t) = \begin{cases} e^{-\mu t} + (1 - e^{-\mu t})\pi_{X\rightarrow X} & \text{if } X=Y \\ (1 - e^{-\mu t})\pi_{X\rightarrow Y} & \text{if } X\neq Y \end{cases}$

$\sim$ know probability of mutating $A\rightarrow G$ in time $t$
Maximum Likelihood Reconstruction

Idea: for a given tree $T$ (with branch lengths), maximize the probability that evolution along $T$ produces observed sequences.

Jukes & Cantor Model

- each base evolves individually
- each base occurs with equal frequency in the genome
- constant rate $\mu$ of mutation
- each base is equally likely to be result of mutation

arc $uv$ of length $t$

$\sim$ flips Poisson distributed (mean $\mu t$)

$$P(\text{no mutation}) = e^{-\mu t}$$

$$P_{X\rightarrow Y}(t) = \begin{cases} 
    e^{-\mu t} + (1 - e^{-\mu t})\pi_{X\rightarrow X} & \text{if } X=Y \\
    (1 - e^{-\mu t})\pi_{X\rightarrow Y} & \text{if } X\neq Y
\end{cases}$$

$\sim$ know probability of mutating $A \rightarrow G$ in time $t$

$\sim$ bottom-up dynamic program

$\sim P(\text{data}|T)$ in $O(|T| \times \#\text{characters})$ time
Maximum Likelihood Reconstruction

Idea: for a given tree $T$ (with branch lengths), maximize the probability that evolution along $T$ produces observed sequences.

**Jukes & Cantor Model**
- each base evolves individually
- each base occurs with equal frequency in the genome
- constant rate $\mu$ of mutation
- each base is equally likely to be result of mutation

arc $uv$ of length $t$
$\rightsquigarrow$ flips poisson distributed (mean $\mu t$)

$$P(\text{no mutation}) = e^{-\mu t}$$

$$P_{X \rightarrow Y}(t) = \begin{cases} 
    e^{-\mu t} + (1 - e^{-\mu t})\pi_{X \rightarrow X} & \text{if } X = Y \\
    (1 - e^{-\mu t})\pi_{X \rightarrow Y} & \text{if } X \neq Y 
\end{cases}$$

$\rightsquigarrow$ know probability of mutating $A \rightarrow G$ in time $t$
$\rightsquigarrow$ bottom-up dynamic program
$\rightsquigarrow$ $P(\text{data}|T)$ in $O(|T| \times \#\text{characters})$ time

theoretically, posterior probability using Bayes’

$$P(T|\text{data}) = P(\text{data}|T) \times P(T) / P(\text{data})$$
Maximum Likelihood Reconstruction

Idea: for a given tree T (with branch lengths), maximize the probability that evolution along T produces observed sequences.

**Jukes & Cantor Model**
- each base evolves individually
- each base occurs with equal frequency in the genome
- constant rate $\mu$ of mutation
- each base is equally likely to be result of mutation

**Generalized Time Reversible Model**
- each base evolves individually
- each base $X$ has a frequency $\varphi_X$ to occur in the genome
- each base-substitution has its own rate $\pi_{X \rightarrow Y}$ of occurrence
Maximum Likelihood Reconstruction

Idea: for a given tree T (with branch lengths), maximize the probability that evolution along T produces observed sequences.

Jukes & Cantor Model
- each base evolves individually
- each base occurs with equal frequency in the genome
- constant rate \( \mu \) of mutation
- each base is equally likely to be result of mutation

Generalized Time Reversible Model
- each base evolves individually
- each base \( X \) has a frequency \( \varphi_X \) to occur in the genome
- each base-substitution has its own rate \( \pi_{X \to Y} \) of occurrence

compute likelihood, given tree & parameters \( \sim O(mn) \) time
Maximum Likelihood Reconstruction

Idea: for a given tree T (with branch lengths), maximize the probability that evolution along T produces observed sequences.

**Jukes & Cantor Model**
- each base evolves individually
- each base occurs with equal frequency in the genome
- constant rate $\mu$ of mutation
- each base is equally likely to be result of mutation

**Generalized Time Reversible Model**
- each base evolves individually
- each base $X$ has a frequency $\phi_X$ to occur in the genome
- each base-substitution has its own rate $\pi_{X \rightarrow Y}$ of occurrence

compute likelihood, given tree & parameters $\sim O(mn)$ time
find "best" tree & parameters $\sim$ NP-hard
Maximum Likelihood Reconstruction

Idea: for a given tree $T$ (with branch lengths), maximize the probability that evolution along $T$ produces observed sequences.

**Jukes & Cantor Model**
- each base evolves individually
- each base occurs with equal frequency in the genome
- constant rate $\mu$ of mutation
- each base is equally likely to be result of mutation

**Generalized Time Reversible Model**
- each base evolves individually
- each base $X$ has a frequency $\phi_X$ to occur in the genome
- each base-substitution has its own rate $\pi_{X \rightarrow Y}$ of occurrence

compute likelihood, given tree $\&$ parameters $\Rightarrow O(mn)$ time
find "best" tree $\&$ parameters $\Rightarrow$ NP-hard
$\Rightarrow$ local search in the tree space (estimate branch-lengths by distances)
ML Reconstruction - Tree Spaces

Observe: a tree and a rearrangement operation span a space

Nearest Neighbor Interchange

change any configuration of 4 (3) "neighboring" subtrees into another
ML Reconstruction - Tree Spaces

Observe: a tree and a rearrangement operation span a space

Nearest Neighbor Interchange

change any configuration of 4 (3) "neighboring" subtrees into another
ML Reconstruction - Tree Spaces

Observe: a tree and a rearrangement operation span a space

Nearest Neighbor Interchange

change any configuration of 4 (3) "neighboring" subtrees into another
ML Reconstruction - Tree Spaces

Observe: a tree and a rearrangement operation span a space

Nearest Neighbor Interchange

change any configuration of 4 (3) "neighboring" subtrees into another

Subtree Prune & Regraft

Break any edge uv & connect v to any edge of the component of u
ML Reconstruction - Tree Spaces

Observe: a tree and a rearrangement operation span a space.

Nearest Neighbor Interchange

change any configuration of 4 (3) "neighboring" subtrees into another

Subtree Prune & Regraft

Break any edge uv and connect v to any edge of the component of u
ML Reconstruction - Tree Spaces

Observe: a tree and a rearrangement operation span a space

Nearest Neighbor Interchange
change any configuration of 4 (3) "neighboring" subtrees into another

Subtree Prune & Regraft
Break any edge uv & connect v to any edge of the component of u
ML Reconstruction - Tree Spaces

Observe: a tree and a rearrangement operation span a space

Nearest Neighbor Interchange

change any configuration of 4 (3) "neighboring" subtrees into another

Subtree Prune & Regraft

Break any edge uv & connect v to any edge of the component of u
ML Reconstruction - Tree Spaces

Observe: a tree and a rearrangement operation span a space

Nearest Neighbor Interchange
change any configuration of 4 (3) "neighboring" subtrees into another

Subtree Prune & Regraft
break any edge uv & connect v to any edge of the component of u
ML Reconstruction - Tree Spaces

Observe: a tree and a rearrangement operation span a space.

Nearest Neighbor Interchange

change any configuration of 4 (3) “neighboring” subtrees into another.

Subtree Prune & Regraft

Break any edge uv & connect v to any edge of the component of u.

Tree Bisection & Reconnection

Break any edge & insert a new reconnecting edge “between” any 2 edges.
ML Reconstruction - Tree Spaces

**Observe**: a tree and a rearrangement operation span a space

**Nearest Neighbor Interchange**

change any configuration of 4 (3) "neighboring" subtrees into another

**Subtree Prune & Regraft**

Break any edge $uv$ &
connect $v$ to any edge of
the component of $u$

**Tree Bisection & Reconnection**

Break any edge &
insert a new
reconnecting edge
"between" any 2 edges
ML Reconstruction - Tree Spaces

Observe: a tree and a rearrangement operation span a space

Nearest Neighbor Interchange

change any configuration of 4 (3) “neighboring” subtrees into another

Subtree Prune & Regraft

Break any edge uv & connect v to any edge of the component of u

Tree Bisection & Reconnection

Break any edge & insert a new reconnecting edge “between” any 2 edges
ML Reconstruction - Tree Spaces

**Observe:** a tree and a rearrangement operation span a space

**Nearest Neighbor Interchange**

change any configuration of 4 (3) "neighboring" subtrees into another

**Subtree Prune & Regraft**

Break any edge $uv \not\in$ connect $v$ to any edge of the component of $u$

**Tree Bisection & Reconnection**

Break any edge $e$ insert a new reconnecting edge "between" any 2 edges
ML Reconstruction - Tree Spaces

Observe: a tree and a rearrangement operation span a space

Nearest Neighbor Interchange
change any configuration of 4 (3) "neighboring" subtrees into another

Subtree Prune & Regraft
Break any edge uv & connect v to any edge of the component of u

Tree Bisection & Reconnection
Break any edge & insert a new reconnecting edge "between" any 2 edges
**Exercise:** turn into (any) caterpillar:

```plaintext
8  9
  0 1
6  2
5  4  3
```

**Exercise:** how are the distances related?
Checking Robustness - Bootstrap Method

**Problem:** Biologists like "strong signal" which is "supported by the data" but edges in trees might be "artefact of optimization"
Checking Robustness - Bootstrap Method

**Problem:** Biologists like "strong signal" which is "supported by the data" but edges in trees might be "artefact of optimization"

⇒ **Bootstrap support** = how "robustly" an edge is supported
Checking Robustness – Bootstrap Method

**Problem:** biologists like “strong signal” which is “supported by the data” but edges in trees might be “artefact of optimization”

~~ bootstrap support = how “robustly” an edge is supported

**Bootstrap Method**

suppose: method X yields tree T from $n \times m$ character-state matrix M
repeat k times the following experiment:

1. draw m columns from M (with repetition)
2. use X to compute $T_i$

Finally, for each branch of T: check how often it occurs in the $T_i$

~~ "bootstrap value" measures robustness ("support") of each branch
Reconstruction by Gene Trees

1. get genomes of multiple species
2. extract genes (annotation)
3. cluster genes in "families" of similar genes
4. within each family, infer a "gene tree" (dist/MP/ML)
5. build a consensus among the gene trees
   ⇝ "species tree"
   (Note: species tree may differ significantly from individual gene trees)
6. reconcile all gene trees with the species tree to learn the evolution of those genes
Reconstruction by Gene Trees

**A Common Method For Reconstructing Trees**

1. get genomes of multiple species
2. extract genes (annotation)
3. cluster genes in "families" of similar genes
4. within each family, infer a "gene tree" (dist/MP/ML)
5. build a consensus among the gene trees $\rightsquigarrow \text{"species tree"}$
   (Note: species tree may differ significantly from individual gene trees)
6. reconcile all gene trees with the species tree to learn the evolution of those genes
Reconstruction by Gene Trees

A Common Method For Reconstructing Trees

1. get genomes of multiple species
2. extract genes (annotation)
3. cluster genes in "families" of similar genes
4. within each family, infer a "gene tree" (dist/MP/ML)
5. build a consensus among the gene trees $\rightsquigarrow$ "species tree"  
   (Note: species tree may differ significantly from individual gene trees)
6. reconcile all gene trees with the species tree to learn the evolution of those genes
Reconstruction by Gene Trees

A Common Method for Reconstructing Trees

1. get genomes of multiple species
2. extract genes (annotation)
3. cluster genes in "families" of similar genes
4. within each family, infer a "gene tree" (dist/MP/ML)
5. build a consensus among the gene trees $\Rightarrow$ "species tree"
   (Note: species tree may differ significantly from individual gene trees)
6. reconcile all gene trees with the species tree to learn the evolution of those genes
Reconstruction by Gene Trees

A Common Method For Reconstructing Trees

1. get genomes of multiple species
2. extract genes (annotation)
3. cluster genes in "families" of similar genes
4. within each family, infer a "gene tree" (dist/MP/ML)
5. build a consensus among the gene trees \( \leadsto \) "species tree"  
   (Note: species tree may differ significantly from individual gene trees)
6. reconcile all gene trees with the species tree to learn the evolution of those genes
Supertrees - "Build" Algorithm

Idea: iteratively refine partition using triplets involving the root
recurse as long as there are $\geq 3$ taxa

Note: always works if trees are compatible/consistent
incompatible? (only 1 component)
- largest compatible subset $\Rightarrow$ NP-hard (even for triplets)
- voting schemes (each tree votes for their clades)
Supertrees - "Build" Algorithm

Idea: iteratively refine partition using triplets involving the root
recursion as long as there are \( \geq 3 \) taxa.

Note: always works if trees are compatible/consistent.

\( \Rightarrow \) largest compatible subset

\(-\) largest compatible subset

\[\text{NP-hard (even for triplets)}\]

\[\text{[Bryant, '97]}\]

\(\text{voting schemes (each tree votes for their clades)}\)
Supertrees - "Build" Algorithm

Idea: iteratively refine partition using triplets involving the root
recurse as long as there are $\geq 3$ taxa

[Aho et al.,'81]

Note: always works if trees are compatible/consistent

- largest compatible subset $\Rightarrow$ NP-hard (even for triplets)
- voting schemes (each tree votes for their clades)

[19 / 38]
Supertrees - "Build" Algorithm

Idea: iteratively refine partition using triplets involving the root & recurse as long as there are \( \geq 3 \) taxa

Note: always works if trees are compatible/consistent

\[ \text{incompatible? (only 1 component)} \rightarrow \text{NP-hard (even for triplets)} \]

\[ \text{Bryant,'97} \]

\[ \text{voting schemes (each tree votes for their clades)} \]
Supertrees - "Build" Algorithm

Idea: iteratively refine partition using triplets involving the root
recurse as long as there are $\geq 3$ taxa

Note: always works if trees are compatible/consistent

- largest compatible subset $\rightarrow$ NP-hard (even for triplets)

Bryant,'97

- voting schemes (each tree votes for their clades)
Supertrees - "Build" Algorithm

Idea: iteratively refine partition using triplets involving the root
recurse as long as there are $\geq 3$ taxa
Supertrees - "Build" Algorithm

Idea: iteratively refine partition using triplets involving the root and recurse as long as there are $\geq 3$ taxa

Note: always works if trees are compatible/consistent

- incompatible? (only 1 component) $\Rightarrow$ largest compatible subset $\Rightarrow$ NP-hard (even for triplets)

[Aho et al.,'81]

[Bryant,'97]

[14x251]Supertrees - "Build" Algorithm

[Aho et al.,'81]

Idea: iteratively refine partition using triplets involving the root and recurse as long as there are $\geq 3$ taxa

Note: always works if trees are compatible/consistent

- incompatible? (only 1 component) $\Rightarrow$ largest compatible subset $\Rightarrow$ NP-hard (even for triplets)

[Aho et al.,'81]

[Bryant,'97]
Supertrees - "Build" Algorithm

Idea: iteratively refine partition using triplets involving the root
recurse as long as there are ≥3 taxa

Note: always works if trees are compatible/consistent
incompatible? (only 1 component)
- largest compatible subset
- voting schemes

[Aho et al., '81]
[Bryant, '97]
Supertrees - "Build" Algorithm

Idea: iteratively refine partition using triplets involving the root
+ recurse as long as there are \( \geq 3 \) taxa

Note: always works if trees are compatible/consistent

- largest compatible subset \( \Rightarrow \) NP-hard (even for triplets)

[Bryant,'97]

- voting schemes (each tree votes for their clades)
Supertrees - "Build" Algorithm

Idea: iteratively refine partition using triplets involving the root
 recurse as long as there are $\geq 3$ taxa

Note: always works if trees are compatible/consistent
 incompatible? (only 1 component)
 $\Rightarrow$ largest compatible subset
 $\Rightarrow$ NP-hard (even for triplets)

[Problems & Voting schemes]

[Bryant '97]
Supertrees - "Build" Algorithm

Idea: iteratively refine partition using triplets involving the root
∧ recurse as long as there are ≥3 taxa

Note: always works if trees are compatible/consistent
- largest compatible subset ⇝ NP-hard (even for triplets)
- voting schemes (each tree votes for their clades)

[Aho et al.,'81]
Supertrees - "Build" Algorithm

Idea: iteratively refine partition using triplets involving the root
+ recurse as long as there are ≥3 taxa

Note: always works if trees are compatible/consistent

Incompatible? (only 1 component)
- largest compatible subset
  \(\leadsto\) NP-hard (even for triplets)
- voting schemes
  (each tree votes for their clades)
Consensi of Non-Agreeing Trees

**Strict Consensus**
- Ochromonas
- Symbiodinium
- Prorocentrum
- Loxodes
- Tetrahymena
- Spirostromum
- Tracheloraphis
- *Euplotes*
- Gruberia

**Consensus Subtree**
- Ochromonas
- Symbiodinium
- Prorocentrum
- Loxodes
- Tetrahymena
- Spirostromum
- *Euplotes*
- Tracheloraphis
- Gruberia

**Majority Consensus**
- Ochromonas
- Symbiodinium
- Prorocentrum
- Loxodes
- Tetrahymena
- Spirostromum
- *Euplotes*
- Tracheloraphis
- Gruberia
- Ochromonas
Reconstruction by Gene Trees

A Common Method For Reconstructing Trees

1. Get genomes of multiple species
2. Extract genes (annotation)
3. Cluster genes in "families" of similar genes
4. Within each family, infer a "gene tree" (dist/MP/ML)
5. Build a consensus among the gene trees $\Rightarrow$ "species tree"
   (Note: species tree may differ significantly from individual gene trees)
6. Reconcile all gene trees with the species tree to learn the evolution of those genes
A Common Method For Reconstructing Trees

1. get genomes of multiple species
2. extract genes (annotation)
3. cluster genes in "families" of similar genes
4. within each family, infer a "gene tree" (dist/MP/ML)
5. build a consensus among the gene trees → "species tree" (Note: species tree may differ significantly from individual gene trees)
6. reconcile all gene trees with the species tree to learn the evolution of those genes
The History of a Gene Family

Recall

gene = "functional element" of DNA, clustered into gene-families

each family yields a tree depicting its history \( \rightsquigarrow \) "gene tree"

consensus of the gene trees \( \rightsquigarrow \) "species tree"

But: what did really happen???

```
Mouse
Dog
Bat
Rat
```

```plaintext
GGAGCTTGAGCCGGAATAGTAGGAAACATCTTTTAAGAAATTATTCTCGAGC
GGAACTCTGAACAGGCTTAGTAGGACCATAAGACTTTTAATTTCGAGC
GGAAATTGGACAGGTTTAGTAGGCACCTAGAATAAGACTTTTAATTTCGAGC
GGAAATTTGACAGCTTAGGCACCTAGAATAAGACTTTTAATTTCGAGC
```

```
Mouse
Bat
Dog
Rat
```

```
Mouse
Rat
Bat
Dog
```

```plaintext
GGAAATTTGACAGCTTAGGCACCTAGAATAAGACTTTTAATTTCGAGC
```
The History of a Gene Family

Recall

gene = "functional element" of DNA, clustered into gene-families.

Each family yields a tree depicting its history ~ "gene tree"
Consensus of the gene trees ~ "species tree"

But: what did really happen???
Reconciliation

Embedding Rules

- Gene tree $G$, species tree $S$
- Mapping $\rho: V(G) \rightarrow V(S)$ s.t. $u \leq_G v \Rightarrow \rho(u) \leq_S \rho(v)$
- $\ell$ is leaf in $G \Rightarrow \rho(\ell)$ "corresponds" to $\ell$ ($a \rightarrow A$, $a' \rightarrow A$, etc.)
- $u \in V(G)$ is called duplication if $\rho(u) = \rho(c)$ for any child $c$ of $u$ in $G$
- All non-leaves, non-duplications of $G$ are called speciations
- Each edge $uv$ of $G$ incurs a loss-cost equal to

$$\left(\#\text{edges in the } \rho(u)-\rho(v)\text{-path in } S\right) - \begin{cases} 1 & \text{if } u \text{ is a speciation} \\ 0 & \text{if } u \text{ is a duplication} \end{cases}$$
Reconciliation

DL-model
- ○ = speciation
- △ = duplication
- × = loss
Reconciliation

Goal: embed gene tree into species tree
(extant genes must map to their species)

Max Likelihood
find most probable embedding
(computationally expensive)

Parsimony
find embedding minimizing
weighted sum of events

DL-model
○ = speciation
△ = duplication
× = loss
Reconciliation

Parsimonious Reconciliation

Input: species tree $S$, gene tree $G$, $\delta, \lambda \in \mathbb{N}$

Task: embed $G$ in $S$, minimizing $\delta \times \#\text{dup} + \lambda \times \#\text{loss}$

Result: LCA-assignment solves this optimally in $O(|S| + |G|)$

DL-model

○ = speciation (0)
△ = duplication ($\delta$)
× = loss ($\lambda$)
Reconciliation

Parsimonious Reconciliation
Input: species tree \( S \), gene tree \( G \), \( \delta, \lambda, \tau \in \mathbb{N} \)
Task: embed \( G \) in \( S \), minimizing \( \delta \times \#\text{dup} + \lambda \times \#\text{loss} \)

Result: LCA-assignment solves this optimally in \( O(|S|+|G|) \)

Exercise
Time
Reconciliation

Parsimonious Reconciliation

Input: species tree $S$, gene tree $G$, $\delta, \lambda, \tau \in \mathbb{N}$

Task: embed $G$ in $S$, minimizing $\delta \times \#\text{dup} + \lambda \times \#\text{loss}$

Result: LCA-assignment solves this optimally in $O(|S|+|G|)$

DTL-model

○ = speciation (0)
△ = duplication ($\delta$)
× = loss ($\lambda$)
□ = transfer ($\tau$)
Reconciliation

**Parsimonious Reconciliation**

*Input*: species tree $S$, gene tree $G$, $\delta$, $\lambda$, $\tau \in \mathbb{N}$

*Task*: embed $G$ in $S$, minimizing $\delta \times \#\text{dup} + \lambda \times \#\text{loss}$

*Result*: LCA-assignment solves this optimally in $O(|S|+|G|)$

*Time constraints* $\Rightarrow$ NP-hard

**DTL-model**

- $\bigcirc$ = speciation (0)
- $\triangle$ = duplication ($\delta$)
- $\times$ = loss ($\lambda$)
- $\diamond$ = transfer ($\tau$)

![Diagram showing the DTL-model with speciation, duplication, loss, and transfer events.]
Parsimonious Reconciliation

Input: species tree $S$, gene tree $G$, $\delta, \lambda, \tau \in \mathbb{N}$

Task: embed $G$ in $S$, minimizing $\delta \times \#dup + \lambda \times \#loss$

Result: LCA-assignment solves this optimally in $O(|S| + |G|)$

$T$ events only between co-existing species $\leadsto$ time constraints $\leadsto$ NP-hard

Idea: dated species tree $\leadsto O(|S|^2|G|)$ [Doyon et al.'10]

DTL-model

- $\circ =$ speciation (0)
- $\triangle =$ duplication ($\delta$)
- $\times =$ loss ($\lambda$)
- $\otimes =$ transfer ($\tau$)
Comparing Phylogenetic Trees

Distance Measures
- Nearest Neighbor Interchange
- Subtree Prune & Regraft
- Tree Bisection & Reconnection
Comparing Phylogenetic Trees

Distance Measures

- Nearest Neighbor Interchange
- Subtree Prune & Regraft
- Tree Bisection & Reconnection
- Robinson-Foulds distance
- quartet/triplet distance
Agreement Forests

**Definition**

A forest $F$ is called **agreement forest** of trees $T_1$ and $T_2$ if $F$ can be obtained from $T_1$ and $T_2$ by removing edges.
Agreement Forests

Definition

A forest $F$ is called agreement forest of trees $T_1$ and $T_2$ if $F$ can be obtained from $T_1$ and $T_2$ by removing edges.
Agreement Forests

Definition

A forest $F$ is called agreement forest of trees $T_1$ and $T_2$ if $F$ can be obtained from $T_1$ and $T_2$ by removing edges.
Agreement Forests

**Definition**

A forest $F$ is called **agreement forest** of trees $T_1$ and $T_2$ if $F$ can be obtained from $T_1$ and $T_2$ by removing edges.
Agreement Forests

Definition
A forest $F$ is called agreement forest of trees $T_1$ and $T_2$ if $F$ can be obtained from $T_1$ and $T_2$ by removing edges.

Theorem [Allen & Steel, ’01]
$\text{TBR-distance}(T_1, T_2) = \#\text{trees in smallest agreement forest} - 1$
NP-hard to compute

Theorem [Bordewich & Semple, ’04]
$\text{rSPR-distance}(T_1, T_2) = \#\text{trees in smallest rooted agreement forest} - 1$
NP-hard to compute
Robinson-Foulds Distance

**Definition**

\[ RF(T_1, T_2) = \#\text{splits/clusters occurring in exactly one of } T_1 \text{ and } T_2 \]

= \( \sum \) edge-contraction distances to a common tree

**Note:** observe relation to NNI: \( RF(T_1, T_2) \leq 2 \ NNI(T_1, T_2) \)

![Diagram of Robinson-Foulds Distance](image)

- trivial splits, \{AB\}|\{C,D,E,F\}, \{EF\}|\{AB,CEF\}, \{CD\}|\{AB,CEF\}
- trivial splits, \{AB\}|\{C,D,E,F\}, \{EF\}|\{AB,CEF\}, \{CD\}|\{AB,CEF\}
Robinson-Foulds Distance

Definition

\[ \text{RF}(T_1, T_2) = \#\text{splits/clusters occurring in exactly one of } T_1 \text{ and } T_2 \]

\[ \quad = \sum \text{edge-contraction distances to a common tree} \]

Note: observe relation to NNI: \( \text{RF}(T_1, T_2) \leq 2 \, \text{NNI}(T_1, T_2) \)

trivial splits, \( \{AB\}|\{C,D,E,F\}, \{EF\}|\{AB,EF\}, \{C,D\}|\{AB,EF\} \)

trivial splits, \( \{AB\}|\{C,D,E,F\}, \{EF\}|\{AB,CD\} \{AB,D\}|\{CEF\} \)
Robinson-Foulds Distance

**Definition**

\[ RF(T_1, T_2) = \text{#splits/clusters occurring in exactly one of } T_1 \text{ and } T_2 \]

\[ = \sum \text{edge-contraction distances to a common tree} \]

**Note:** observe relation to NNI: \( RF(T_1, T_2) \leq 2 \cdot \text{NNI}(T_1, T_2) \)

(C) Leonardo de Oliveira Martins
Robinson-Foulds Distance

**Definition**

\[ RF(T_1, T_2) = \# \text{splits/clusters occurring in exactly one of } T_1 \text{ and } T_2 = \sum \text{edge-contraction distances to a common tree} \]

**Note:** observe relation to NNI: \( RF(T_1, T_2) \leq 2 \text{ NNI}(T_1, T_2) \)

---

**Day’s Algorithm (common clusters in } O(n)\)\)

[Day’85]

1. relabel all leaves s.t. leaves continuous in \( T_1 \)
2. each vertex in \( T_1 \) knows:
   - \( L \): smallest leaf in cluster
   - \( R \): largest leaf in cluster
   - \( \sim T_1 \)‘s clusters in table \([L, R]\) \((O(n) \text{ sparse set/perfect hashing})\)
3. each vertex in \( T_2 \) knows \( L \neq R \neq \) size \( N \) of its cluster
4. each vertex in \( T_2 \) only checks for \([L, R]\) if \( R - L = N - 1 \)
   (lookup in \( T_1 \)’s table in \( O(1) \) (average) time)
Robinson-Foulds Distance

Day’s Algorithm (common clusters in $O(n)$)

1. relabel all leaves s.t. leaves continuous in $T_1$
2. each vertex in $T_1$ knows:
   - $L$: smallest leaf in cluster
   - $R$: largest leaf in cluster
   $\rightsquigarrow$ $T_1$’s clusters in table $[L, R]$ ($O(n)$ sparse set/perfect hashing)
3. each vertex in $T_2$ knows $L \neq R \neq$ size $N$ of its cluster
4. each vertex in $T_2$ only checks for $[L, R]$ if $R - L = N - 1$
   (lookup in $T_1$’s table in $O(1)$ (average) time)
Robinson-Foulds Distance

Day’s Algorithm (common clusters in $O(n)$)

1. relabel all leaves s.t. leaves continuous in $T_1$
2. each vertex in $T_1$ knows:
   - $L$: smallest leaf in cluster
   - $R$: largest leaf in cluster
   - $T_1$’s clusters in table $[L, R]$ ($O(n)$ sparse set/perfect hashing)
3. each vertex in $T_2$ knows $L \neq R \neq$ size $N$ of its cluster
4. each vertex in $T_2$ only checks for $[L, R]$ if $R - L = N - 1$
   (lookup in $T_1$’s table in $O(1)$ (average) time)

[Day’85]
Robinson-Foulds Distance

Day’s Algorithm (common clusters in $O(n)$)

1. relabel all leaves s.t. leaves continuous in $T_1$
2. each vertex in $T_1$ knows:
   - $L$: smallest leaf in cluster
   - $R$: largest leaf in cluster
   - $\Rightarrow$ $T_1$’s clusters in table $[L, R]$ ($O(n)$ sparse set/perfect hashing)
3. each vertex in $T_2$ knows $L \neq R \neq$ size $N$ of its cluster
4. each vertex in $T_2$ only checks for $[L, R]$ if $R - L = N - 1$ (lookup in $T_1$’s table in $O(1)$ (average) time)
Robinson-Foulds Distance

Day’s Algorithm (common clusters in $O(n)$)

1. relabel all leaves s.t. leaves continuous in $T_1$
2. each vertex in $T_1$ knows:
   - $L$: smallest leaf in cluster
   - $R$: largest leaf in cluster
   
   $\rightarrow$ $T_1$’s clusters in table $[L, R]$ ($O(n)$ sparse set/perfect hashing)
3. each vertex in $T_2$ knows $L \neq R \neq$ size $N$ of its cluster
4. each vertex in $T_2$ only checks for $[L, R]$ if $R - L = N - 1$
   (lookup in $T_1$’s table in $O(1)$ (average) time)
Quartet/Triplet Distance

\[ Q/T(T_1, T_2) = \#\text{quartets/triplets occur. in exactly one of } T_1 \text{ and } T_2 \]
Quartet/Triplet Distance

$Q/T(T_1, T_2) = \#\text{quartets/triplets occur. in exactly one of } T_1 \text{ and } T_2$

computing Q-distance (binary trees) [Bryant et al.'00**]

Idea: each edge uv gives rise to set of quartets $\leadsto$ intersect quartet set of each edge in $T_1$ with each edge in $T_2 \leadsto O(n^2)$ intersections
Quartet/Triplet Distance

\[
Q/T(T_1,T_2) = \#\text{quartets/triplets occur. in exactly one of } T_1 \text{ and } T_2
\]

computing Q-distance (binary trees)

Idea: each edge \( uv \) gives rise to set of quartets \( \mapsto \) intersect quartet set of each edge in \( T_1 \) with each edge in \( T_2 \) \( \mapsto O(n^2) \) intersections

Idea: only need number of common quartets (sizes of intersections)

[Bryant et al.'00**]
Quartet/Triplet Distance

\[ Q/T(T_1, T_2) = \#\text{quartets/triplets occur in exactly one of } T_1 \text{ and } T_2 \]

computing Q-distance (binary trees) [Bryant et al.’00**]

Idea: each edge uv gives rise to set of quartets \( \sim \) intersect quartet set of each edge in \( T_1 \) with each edge in \( T_2 \) \( \sim O(n^2) \) intersections

Idea: only need number of common quartets (sizes of intersections)

But: must avoid double counting

\( \sim \) each edge in two directions \( \neq \) count only quartets touching the head

1. quartet \( AB|CD \) "Belongs" to edge \( u \rightarrow v \) if \( u \rightarrow v \) splits \( AB|CD \) \( \neq \) \( v \) touches \( AB\)-path \( \sim \) each quartet is owned exactly twice

\[
\#\text{common } Q(vu, ij) = \left( |c(uv) \cap c(ji)|^2 \right) \times \left( |c(ut) \cap c(jk)| \times |c(uw) \cap c(jB)| + |c(uw) \cap c(jB)| \times |c(ut) \cap c(jk)| \right)
\]
Quartet/Triplet Distance

\[ Q/T(T_1, T_2) = \#\text{quartets/triplets occur in exactly one of } T_1 \text{ and } T_2 \]

computing Q-distance (binary trees) [Bryant et al.'00**]

Idea: each edge uv gives rise to set of quartets \( \leadsto \) intersect quartet set of each edge in \( T_1 \) with each edge in \( T_2 \) \( \leadsto O(n^2) \) intersections

Idea: only need number of common quartets (sizes of intersections)

But: must avoid double counting

\( \leadsto \) each edge in two directions \( \neq \) count only quartets touching the head

1. quartet \( AB|CD \) "Belongs" to edge \( u \rightarrow v \) if \( u \rightarrow v \) splits \( AB|CD \) \( \neq \)
   \( v \) touches \( AB \)-path \( \leadsto \) each quartet is owned exactly twice

\[
\begin{align*}
\text{u \rightarrow v: } & YZ|*, YZ|* \\
\text{c(vu) = } & \{A, B, C, D\} \\
\text{#commonQ(vu,ij) = } & \left( |c(uv) \cap c(ji)|^2 \right) \times \left( |c(ut) \cap c(jk)| \times |c(uw) \cap c(jB)| + |c(uw) \cap c(jB)| \times |c(ut) \cap c(jk)| \right)
\end{align*}
\]
Quartet/Triplet Distance

\[ Q/T(T_1, T_2) = \#\text{quartets/triplets occur in exactly one of } T_1 \text{ and } T_2 \]

computing Q-distance (binary trees) [Bryant et al.'00**]

Idea: each edge uv gives rise to set of quartets \( \leadsto \) intersect quartet set of each edge in \( T_1 \) with each edge in \( T_2 \) \( \leadsto O(n^2) \) intersections

Idea: only need number of common quartets (sizes of intersections)

But: must avoid double counting

\( \leadsto \) each edge in two directions \( \neq \) count only quartets touching the head

1. quartet AB|CD "belongs" to edge \( u \rightarrow v \) if \( u \rightarrow v \) splits AB|CD \( \neq \)
   - \( v \) touches AB-path \( \leadsto \) each quartet is owned exactly twice

\( u \rightarrow v: \ YZ|*, YZ|* \)

\( v \rightarrow u: \ AC|*, AD|*, BC|*, BD|* \)
Quartet/Triplet Distance

\[ Q/T(T_1, T_2) = \#\text{quartets/triplets occur in exactly one of } T_1 \text{ and } T_2 \]

computing Q-distance (binary trees) [Bryant et al.’00**]

Idea: each edge \( uv \) gives rise to set of quartets \( \rightsquigarrow \) intersect quartet set of each edge in \( T_1 \) with each edge in \( T_2 \) \( \rightsquigarrow O(n^2) \) intersections

Idea: only need number of common quartets (sizes of intersections)

But: must avoid double counting

\( \rightsquigarrow \) each edge in two directions \( \neq \) count only quartets touching the head

1. quartet \( AB|CD \) "belongs" to edge \( u \rightarrow v \) if \( u \rightarrow v \) splits \( AB|CD \) \( \neq \)

   \( v \) touches \( AB \)-path \( \rightsquigarrow \) each quartet is owned exactly twice

\[
c(vu) = \{AB, C, D\}
\]

\[
\text{u} \rightarrow \text{v}: YZ|*, YZ|*
\]

\[
v \rightarrow \text{u}: AC|*, AD|*, BC|*, BD|*
\]
Quartet/Triplet Distance

\[ Q/T(T_1, T_2) = \# \text{quartets/triplets occur in exactly one of } T_1 \text{ and } T_2 \]

computing Q-distance (binary trees)  

[Bryant et al.’00**]

Idea: each edge uv gives rise to set of quartets \( \Rightarrow \) intersect quartet set of each edge in \( T_1 \) with each edge in \( T_2 \) \( \Rightarrow O(n^2) \) intersections

Idea: only need number of common quartets (sizes of intersections)

But: must avoid double counting

\( \Rightarrow \) each edge in two directions \( \neq \) count only quartets touching the head

1. quartet \( AB|CD \) "belongs" to edge \( u \rightarrow v \) if \( u \rightarrow v \) splits \( AB|CD \)

\( v \) touches \( AB\)-path \( \Rightarrow \) each quartet is owned exactly twice

2. \( \forall e_1 \in T_1 \neq e_2 \in T_2: \# \text{common quart. from sizes of clade-intersec.} \)

\[ #\text{commonQ}(vu,ij) = \left( |c(uv) \cap c(ji)| \right)^2 \times \]

\( (|c(ut) \cap c(jk)| \times |c(uw) \cap c(jB)| + |c(uw) \cap c(jB)| \times |c(ut) \cap c(jk)|) \)

\[ \begin{array}{c}
D \quad t \\
C \\
B \quad w \\
A \\
\hline
\end{array} \hspace{1cm} \begin{array}{c}
Z \\
Y \\
X \\
A \\
\hline
\end{array} \hspace{1cm} \begin{array}{c}
Z \\
C \\
X \\
A \\
\hline
B \quad j \\
Y \\
k \quad D \\
\hline
\end{array} \]
Quartet/Triplet Distance

\[ Q/T(T_1, T_2) = \#\text{quartets/triplets occur in exactly one of } T_1 \text{ and } T_2 \]

computing \textbf{Q-distance (binary trees)} \[ [\text{Bryant et al.'00}**] \]

\textbf{Idea:} each edge uv gives rise to set of quartets \( \implies \) intersect quartet set of each edge in \( T_1 \) with each edge in \( T_2 \) \( \implies O(n^2) \) intersections

\textbf{Idea:} only need number of common quartets (sizes of intersections)

\textbf{But:} must avoid double counting

\( \implies \) each edge in two directions \( \neq \) count only quartets touching the head

1. quartet AB|CD "belongs" to edge \( u \to v \) if \( u \to v \) splits \( AB|CD \) \( \neq \)

2. \( \forall e_1 \in T_1 \neq e_2 \in T_2 \): \#common quart. from sizes of clade-intersec.

3. sizes of all intersections precomputed bottom-up in \( O(n^2) \) time

\[ |c(vu) \cap c_2| = |c(uw) \cap c_2| + |c(ut) \cap c_2| \]
Quartet/Triplet Distance

\[ Q/T(T_1, T_2) = \#\text{quartets/triplets occur in exactly one of } T_1 \text{ and } T_2 \]

computing Q-distance (binary trees) [Bryant et al.'00**]

Idea: each edge uv gives rise to set of quartets \( \mapsto \) intersect quartet set of each edge in \( T_1 \) with each edge in \( T_2 \) \( \mapsto O(n^2) \) intersections

Idea: only need number of common quartets (sizes of intersections)

But: must avoid double counting

\( \mapsto \) each edge in two directions \( \neq \) count only quartets touching the head

1. quartet \( AB|CD \) "belongs" to edge \( u \rightarrow v \) if \( u \rightarrow v \) splits \( AB|CD \) \( \neq \)

\( v \) touches \( AB \)-path \( \mapsto \) each quartet is owned exactly twice

2. \( \forall e_1 \in T_1 \neq e_2 \in T_2: \#\text{common quart. from sizes of clade-intersec.} \)

3. sizes of all intersections precomputed bottom-up in \( O(n^2) \) time

State of the Art

count conflict quartets/triplets \( \mapsto O(n \log n) \) time [Brodal et al.'13]

enumerate conflict quartets \( \mapsto O(n^2 + d) \) [Bryant et al.'00]

enumerate conflict triplets \( \mapsto O(n + d) \) [Weller'17]
Phylogenetic Networks

Observation
Trees cannot capture hybridization
Phylogenetic Networks

Observation
Trees cannot capture hybridization $\Rightarrow$ phylogenetic network
Phylogenetic Networks

Observation
Trees cannot capture hybridization \( \rightarrow \) phylogenetic network

Definition
Evolutionary network \( N = \) rooted DAG, leaves labeled (taxa)
Reticulations \( R = \) vertices of in-degree \( \geq 2 \)
Binary = all inner vertices degree 3
Block = component without cut-vertex
Display \( T = \) subdivision of \( T \) is a subgraph
Phylogenetic Networks

Observation
Trees cannot capture hybridization \( \rightarrow \) phylogenetic network

Definition

- **evolutionary network** \( N = \) rooted DAG, leaves labeled (taxa)
- **reticulations** \( R = \) vertices of in-degree \( \geq 2 \)
- **binary** = all inner vertices degree 3
- **block** = component without cut-vertex
- **display** \( T = \) subdivision of \( T \) is a subgraph
Split Networks

split = bipartition of set of taxa
splits $A|B \neq X|Y$ incompatible if both $A \cap B$ intersect both $X \cap Y$

Convex Hull Algorithm [Holland et al., ’04]

Idea 1: consider binary character as split
Idea 2: duplicate nodes of convex hull
Split Networks

\[ \text{split} = \text{bipartition of set of taxa} \]

splits \( A|B \) \& \( X|Y \) incompatible if both \( A \cap B \) intersect both \( X \cap Y \)

Convex Hull Algorithm [Holland et al.,'04]

Idea 1: consider binary character as split
Idea 2: duplicate nodes of convex hull
Split Networks

split = bipartition of set of taxa

splits $A \mid B \neq X \mid Y$ incompatible if both $A \neq B$ intersect both $X \neq Y$

Convex Hull Algorithm [Holland et al., '04]

Idea 1: consider binary character as split
Idea 2: duplicate nodes of convex hull
Split Networks

split = bipartition of set of taxa
splits $A \cup B \neq X \cup Y$ incompatible if both $A \cap B$ intersect both $X \cap Y$

Convex Hull Algorithm [Holland et al.,'04]

Idea 1: consider binary character as split
Idea 2: duplicate nodes of convex hull
Split Networks

split = bipartition of set of taxa
splits A|B & X|Y incompatible if both A ∩ B intersect both X ∩ Y

Convex Hull Algorithm [Holland et al., '04]

Idea 1: consider binary character as split
Idea 2: duplicate nodes of convex hull
Split Networks

split = bipartition of set of taxa
splits $A|B \neq X|Y$ incompatible if both $A \neq B$ intersect both $X \neq Y$

Convex Hull Algorithm [Holland et al.,'04]

Idea 1: consider binary character as split
Idea 2: duplicate nodes of convex hull
Split Networks

split = bipartition of set of taxa
splits $A \cup B \neq X \cup Y$ incompatible if both $A \cap B$ intersect both $X \cap Y$

Convex Hull Algorithm [Holland et al., '04]

Idea 1: consider binary character as split
Idea 2: duplicate nodes of convex hull
Split Networks

split = bipartition of set of taxa

splits A|B \& X|Y incompatible if both A \& B intersect both X \& Y

Convex Hull Algorithm [Holland et al., '04]

Idea 1: consider binary character as split
Idea 2: duplicate nodes of convex hull
Split Networks

\[
\text{split} = \text{bipartition of set of taxa}
\]

splits \(A \mid B \neq X \mid Y\) incompatible if both \(A \cap B\) intersect both \(X \cap Y\)

**Convex Hull Algorithm** [Holland et al.,'04]

**Idea 1:** consider binary character as split

**Idea 2:** duplicate nodes of convex hull
Split Networks

\[ split = \text{bipartition of set of taxa} \]

splits \( A|B \neq X|Y \) incompatible if both \( A \cap B \) intersect both \( X \cap Y \)

Convex Hull Algorithm \([\text{Holland et al.},'04]\)

**c.f.: Neighbor Net** \([\text{Bryant \\& Moulton},'03]\)

(for circular splits)
Consensus Network from Partial Trees

Super-networks: Z-closure

Build network from splits (from trees)

\[
\begin{align*}
A & \geq X \\
B & \geq Y
\end{align*}
\]

splits \(A|B \neq X|Y \leadsto A|BY \neq AX|Y\)

condition: \(A \cap X \neq \emptyset, B \cap X \neq \emptyset, B \cap Y \neq \emptyset, A \cap Y = \emptyset\)
Consensus Network from Partial Trees

Super-networks: Z-closure

Build network from splits (from trees)

\[
\begin{align*}
A & \leq X \\
B & \geq Y
\end{align*}
\]

splits \( A\mid B \neq X\mid Y \Rightarrow A\mid BY \neq A\times Y \)

condition: \( A \cap X \neq \emptyset, B \cap X \neq \emptyset, B \cap Y \neq \emptyset, A \cap Y = \emptyset \)

Strategy

1. list all splits of all input trees
2. extend splits to full taxa using "Z-closure"
3. build split network
Recombination Networks

Observation

rooted network: cluster of $u \subseteq$ cluster of $v \iff u \leq v$

\Rightarrow rooted network is hasse diagram of its clusters
Recombination Networks

Observation

rooted network: cluster of $u \subseteq$ cluster of $v \iff u \leq v$

\[ \Rightarrow \text{rooted network is Hasse diagram of its clusters} \]

Example

\{a,b,c,d\}, \{c,d,e,f,g,h\}, \{c,d,e,f,g\}, \{e,f,g,h\}, \{c,d,e\}, \{e,f,g\}, \{a,b\}, \{c,d\}, \{f,g\}

Exercise Time
Recombination Networks

Observation

rooted network: cluster of $u \subseteq$ cluster of $v \iff u \leq v$

~ rooted network is hasse diagram of its clusters

Example

\{a,b,c,d\}, \{c,d,e,f,g,h\}, \{c,d,e,f,g\}, \{e,f,g,h\}, \{c,d,e\}, \{e,f,g\}, \{a,b\}, \{c,d\}, \{f,g\}
Recombination Networks

Observation
rooted network: cluster of $u \subseteq$ cluster of $v \iff u \leq v$
$\iff$ rooted network is hasse diagram of its clusters

Example
\{a,b,c,d\}, \{c,d,e,f,g,h\}, \{c,d,e,f,g\}, \{e,f,g,h\}, \{c,d,e\}, \{e,f,g\}, \{a,b\}, \{c,d\}, \{f,g\}
Recombination Networks

Observation

rooted network: cluster of $u \subseteq$ cluster of $v \iff u \leq v$

$\implies$ rooted network is hasse diagram of its clusters

Example

$\{a, b, c, d\}, \{c, d, e, f, g, h\}, \{c, d, e, f, g\}, \{e, f, g, h\}, \{c, d, e\}, \{e, f, g\}, \{a, b\}, \{c, d\}, \{f, g\}$

c.f. “cluster popping” [Huson & Rupp,'08]
Recombination Networks

**Observation**

rooted network: cluster of $u \subseteq$ cluster of $v \iff u \leq v$

$\Rightarrow$ rooted network is hasse diagram of its clusters

**Problem:** may produce more reticulations than necessary
Recombination Networks

**Observation**

rooted network: cluster of \( u \subseteq \) cluster of \( v \) \( \iff u \leq v \)

\( \leadsto \) rooted network is **hasse diagram** of its clusters

**Problem:** may produce more reticulations than necessary

**Hybridization Number**

**Input:** set of trees \( T \), int \( k \)

**Question:** Is there a network with \( \leq k \) reticulations displaying all trees in \( T \)?
Recombination Networks

**Observation**
rooted network: cluster of $u \subseteq$ cluster of $v \iff u \leq v$
$\Rightarrow$ rooted network is hasse diagram of its clusters

**Problem**: may produce more reticulations than necessary

**Hybridization Number**
Input: set of trees $T$, int $k$
Question: Is there a network with $\leq k$ reticulations displaying all trees in $T$?
$\Rightarrow$ NP-hard for 2 trees

[Baroni et al.,'05]
[Bordewich & Semple,'07]
Recombination Networks

Observation

rooted network: cluster of $u \subseteq$ cluster of $v \Leftrightarrow u \leq v$

$\Rightarrow$ rooted network is hasse diagram of its clusters

Problem: may produce more reticulations than necessary

Hybridization Number

Input: set of trees $T$, int $k$

Question: Is there a network with $\leq k$ reticulations displaying all trees in $T$?

$\Rightarrow$ NP-hard for 2 trees [Bordewich & Semple,'07]

Note: $HN(T_1,T_2) = \text{max. acyclic agreement forest} - 1$ [Baroni et al.,'05]
Recombination Networks

**Observation**

rooted network: cluster of \( u \subseteq \) cluster of \( v \) \( \iff u \leq v \)

\( \leadsto \) rooted network is *Hasse diagram* of its clusters

**Problem:** may produce more reticulations than necessary

**Hybridization Number**

**Input:** set of trees \( T \), int \( k \)

**Question:** Is there a network with \( \leq k \) reticulations displaying all trees in \( T \)?

\( \leadsto \) NP-hard for 2 trees

**Note:** \( HN(T_1, T_2) = \max \) acyclic agreement forest - 1

\[ \text{[Bordewich & Semple,'07]} \]

\[ \text{[Baroni et al.,'05]} \]
Recombination Networks

**Observation**

rooted network: cluster of $u \subseteq$ cluster of $v \leftrightarrow u \leq v$

$\Rightarrow$ rooted network is hasse diagram of its clusters

**Problem:** may produce more reticulations than necessary

**Hybridization Number**

**Input:** set of trees $T$, int $k$

**Question:** Is there a network with $\leq k$ reticulations displaying all trees in $T$?

$\Rightarrow$ NP-hard for 2 trees

**Note:** $HN(T_1, T_2) = \max$ acyclic agreement forest - 1
Recombination Networks

Observation
rooted network: cluster of $u \subseteq$ cluster of $v \iff u \leq v$
$\Rightarrow$ rooted network is hasse diagram of its clusters

Problem: may produce more reticulations than necessary

Hybridization Number
Input: set of trees $T$, int $k$

Question: Is there a network with $\leq k$ reticulations displaying all trees in $T$?

$\Rightarrow$ NP-hard for 2 trees
[Baroni et al.,'05]

Note: $HN(T_1,T_2) = \max$ acyclic agreement forest - 1
[Bordewich & Semple,'07]
Recombination Networks

**Observation**

rooted network: cluster of u ⊆ cluster of v ⇔ u ≤ v
⇒ rooted network is hasse diagram of its clusters

**Problem:** may produce more reticulations than necessary

**Hybridization Number**

Input: set of trees T, int k

Question: Is there a network with ≤ k reticulations displaying all trees in T?

⇒ NP-hard for 2 trees

**Note:** \( \text{HN}(T_1, T_2) = \text{max. acyclic agreement forest} - 1 \)

[Bordewich & Semple,'07] [Baroni et al.,'05]
Recombination Networks

Observation
rooted network: cluster of \( u \subseteq \) cluster of \( v \) \( \iff u \leq v \)
\( \leadsto \) rooted network is Hasse diagram of its clusters

Problem: may produce more reticulations than necessary

Hybridization Number
Input: set of trees \( T \), int \( k \)

Question: Is there a network with \( \leq k \) reticulations displaying all trees in \( T \)?

\( \leadsto \) NP-hard for 2 trees

Note: \( HN(T_1, T_2) = \text{max. acyclic agreement forest} - 1 \)
Observation

A network may display up to $2^{|R|}$ trees.
Observation

A network may display up to $2^{\left| R \right|}$ trees.
Networks Display Trees

Observation

A network may display up to $2^{|R|}$ trees. But: how to decide if a given tree is displayed?
Networks Display Trees

Tree Containment

Input: a network $N$, a tree $T$

Question: Does $N$ display $T$?
Networks Display Trees

Tree Containment
Input: a network $N$, a tree $T$

Question: Does $N$ display $T$?

$\Rightarrow$ NP-hard (from Disjoint Paths) [Kanj et al.'08]

Note: linear time on reticulation visible $N$ [Gunawan, '18; Weller, '18]
Networks Display Trees

Tree Containment
Input: a network $N$, a tree $T$

Question: Does $N$ display $T$?

$\Rightarrow$ NP-hard (from Disjoint Paths) [Kanj et al.'08]

Note: linear time on reticulation visible $N$ [Gunawan,'18][Weller,'18]
Networks Display Trees

Tree Containment

**Input:** a network $N$, a tree $T$

**Question:** Does $N$ display $T$?

$\Rightarrow$ NP-hard (from Disjoint Paths) [Kanj et al.'08]

**Note:** linear time on reticulation visible $N$ [Gunawan,'18][Weller,'18]
Building Block I: Multi-labeled Trees

Lemma ($k$-labeled trees)

Let $v \in T$: $M(v) =$ lowest vertices $u$ s.t. $N_u$ displays $T_v$.

Let $v_1, v_2$ be the children of $v$. Then, $M(v) = \min_{N \bigcup u, w \in M(v_i)} LCA(N(u), w) \setminus \bigcup_i M(v_i)$.

$\Rightarrow |M(v)| \leq k$ & we compute $M$ in $O(|N| \cdot k^2)$ time.

Corollary We get highest $v$ s.t. $N$ displays $T_v$ in $O(|N| \cdot k^2)$ time.

These $T_v$ are leaf-disjoint.
**Lemma (k-labeled trees)**

\[ v \in T: M(v) = \text{lowest vertices } u \text{ s.t. } N_u \text{ displays } T_v. \]

Let \( v_1, v_2 \) be the children of \( v \). Then,

\[
M(v) = \min_{u,w \in M(v_i)} \left( \bigcup_{u,w \in M(v_i)} \text{LCA}_N(u, w) \setminus \bigcup_{i} M(v_i) \right)
\]
Lemma (\(k\)-labeled trees)

Let \(v \in T\): \(M(v) = \) lowest vertices \(u\) s.t. \(N_u\) displays \(T_v\).

Let \(v_1, v_2\) be the children of \(v\). Then,

\[
M(v) = \min_{N} \left( \bigcup_{u, w \in M(v_i)} \text{LCA}_N(u, w) \setminus \bigcup_{i} M(v_i) \right)
\]
**Lemma** ($k$-labeled trees)

$v \in T$: $M(v) = \{u \mid u \text{ s.t. } N_u \text{ displays } T_v\}$. Let $v_1, v_2$ be the children of $v$. Then,

$$M(v) = \min_{N} \left( \bigcup_{u, w \in M(v_i)} \text{LCA}_N(u, w) \setminus \bigcup_{i} M(v_i) \right)$$

These $T_v$ are leaf-disjoint.
Lemma \((k\text{-labeled trees})\)

\(v \in T: M(v) = \text{lowest vertices } u \text{ s.t. } N_u \text{ displays } T_v.\)

Let \(v_1, v_2\) be the children of \(v\). Then,

\[
M(v) = \min_{v_i} \left( \bigcup_{u, w \in M(v_i)} \text{LCA}_N(u, w) \setminus \bigcup_i M(v_i) \right)
\]
Lemma \((k\text{-labeled trees})\)

\(v \in T: M(v) = \text{lowest vertices } u \text{ s.t. } N_u \text{ displays } T_v.\) Let \(v_1, v_2\) be the children of \(v\). Then,

\[
M(v) = \min_N \left( \bigcup_{u, w \in M(v_i)} LCA_N(u, w) \setminus \bigcup_i M(v_i) \right)
\]
**Lemma (k-labeled trees)**

$v \in T$: $M(v) =$ lowest vertices $u$ s.t. $N_u$ displays $T_v$.

Let $v_1, v_2$ be the children of $v$. Then,

$$M(v) = \min_{N} \left( \bigcup_{u, w \in M(v_i)} \text{LCA}_N(u, w) \setminus \bigcup_i M(v_i) \right)$$

$\Rightarrow |M(v)| \leq k \iff$ we compute $M$ in $O(|N| \cdot k^2)$ time.
**Building Block 1: Multi-labeled Trees**

**Lemma** \((k\text{-labeled trees})\)

\(v \in T: M(v) = \text{lowest vertices } u \text{ s.t. } N_u \text{ displays } T_v\)

Let \(v_1, v_2\) be the children of \(v\). Then,

\[
M(v) = \min_N \left( \bigcup_{u,w \in M(v_i)} \text{LCA}_N(u,w) \right)
\]

\(\Rightarrow |M(v)| \leq k \quad \text{we compute } M \text{ in } O(|N| \cdot k^2) \text{ time.}\)

**Corollary**

We get highest \(v\) s.t. \(N\) displays \(T_v\) in \(O(|N| \cdot k^2)\) time.

These \(T_v\) are leaf-disjoint.
Building Block II: Component DAGs

**Observation**  

\[ N - R = \text{forest} \quad \sim \quad N = \text{DAG of tree components} + R \]

[Gunawan et al., '16]
Building Block II: Component DAGs

Observation

\[ N - R = \text{forest} \quad \sim \quad N = \text{DAG of tree components} + R \]

[Gunawan et al., '16]
Observation

\( N - R = \text{forest} \quad \sim \quad N = \text{DAG of tree components} + R \)

[Gunawan et al., ’16]
Building Block II: Component DAGs

**Observation**

\[ N - R = \text{forest} \quad \sim \quad N = \text{DAG of tree components} + R \]

[Gunawan et al., '16]
Building Block II: Component DAGs

Observation

\[ N - R = \text{forest} \quad \leadsto \quad N = \text{DAG of tree components} + R \]
Building Block II: Component DAGs

Observation

\[ N - R = \text{forest} \sim N = \text{DAG of tree components} + R \]

[Gunawan et al., '16]
Combining the Building Blocks

Network

Recall: Can get highest TV displayed by NR in O(|NR| · k^2)

Definition r is stable on ⇔ all root-paths contain r

Reduction Rule Root r of lowest tree-component NR in N is stable on.

⇝ collapse NR & the highest TV with ∈ TV that NR displays.

Problem Which TV to choose?

Choose the TV containing the leaves that are accessible only via r!

Theorem solve TC in O(|N|) if (a) max. indegree (reti. chain) is constant & (b) all tree-component-roots are stable.

Corollary solve TC in O(|N|) on (a) reticulation visible (all reti. stable) & (b) nearly-stable (unstable ⇒ stable parent)

Corollary TC can be solved in O(|N|^3) time on networks N in which all tree-vertices with reticulation parents are stable.
Combining the Building Blocks

Recall:

\[ \text{Can get highest } \mathcal{T}_r \text{ displayed by } N_r \text{ in } O(|N_r| \cdot k^2) \]

**Definition**

\( r \) is stable on \( \iff \) all root-paths contain \( r \)

**Reduction Rule**

Root \( r \) of lowest tree-component \( N_r \) in \( N \) is stable on.

\( \Rightarrow \) collapse \( N_r \) & the highest \( \mathcal{T}_v \) with \( \in \mathcal{T}_v \) that \( N_r \) displays.

**Problem**

Which \( \mathcal{T}_v \) to choose?

Choose the \( \mathcal{T}_v \) containing the leaves that are accessible only via \( r \).

**Theorem**

Solve TC in \( O(|N|) \) if

(a) \( \max \text{ indegree (reti. chain)} \) is constant &

(b) all tree-component-roots are stable.

**Corollary**

Solve TC in \( O(|N|) \) on

(a) reticulation visible (all reti. stable) &

(b) nearly-stable (unstable \( \Rightarrow \) stable parent).

**Corollary**

TC can be solved in \( O(|N|^3) \) time on networks \( N \) in which all tree-vertices with reticulation parents are stable.
Combining the Building Blocks

Network \( \Rightarrow \) Tree

Recall: Can get highest \( T_v \) displayed by \( N_r \) in \( O(|N_r| \cdot k^2) \)

Definition \( r \) is stable on \( \iff \) all root-paths contain \( r \)

Reduction Rule
Root \( r \) of lowest tree-component \( N_r \) in \( N \) is stable on \( \Rightarrow \) collapse \( N_r \) & the highest \( T_v \) with \( \epsilon \in T_v \) that \( N_r \) displays.

Problem Which \( T_v \) to choose?
Choose the \( T_v \) containing the leaves that are accessible only via \( r! \)

Theorem solve TC in \( O(|N|) \) if
(a) max. indegree (reti. chain) is constant &
(b) all tree-component-roots are stable.

Corollary solve TC in \( O(|N|) \) on
(a) reticulation visible (all reti. stable) &
(b) nearly-stable (unstable \( \Rightarrow \) stable parent)

Corollary TC can be solved in \( O(|N|^3) \) time on networks \( N \) in which all tree-vertices with reticulation parents are stable.

35 / 38
Combining the Building Blocks

Network

Tree

Recall:
Can get highest $T_v$ displayed by $N_r$ in $O(|N_r| \cdot k^2)$
Combining the Building Blocks

**Network**

**Tree**

**Recall:**

Can get highest $T_v$ displayed by $N_r$ in $O(|N_r| \cdot k^2)$

**Problem**

Which $T_v$ to choose?
Combining the Building Blocks

**Recall:**

Can get highest $T_v$ displayed by $N_r$ in $O(|N_r| \cdot k^2)$

**Problem**

Which $T_v$ to choose?
Combining the Building Blocks

Recall:
Can get highest $T_v$ displayed by $N_r$ in $O(|N_r| \cdot k^2)$

Problem
Which $T_v$ to choose?
Combining the Building Blocks

**Recall:**
Can get highest $T_v$ displayed by $N_r$ in $O(|N_r| \cdot k^2)$

Choose the $T_v$ containing the leaves that are accessible only via $r$!
Combining the Building Blocks

**Recall:**
Can get highest $T_v$ displayed by $N_r$ in $O(|N_r| \cdot k^2)$.

**Definition**
$r$ is stable on $N_r$ if all root-paths contain $r$.

Choose the $T_v$ containing the leaves that are accessible only via $r$!
Combining the Building Blocks

**Network**

![Network Diagram]

**Tree**

![Tree Diagram]

**Recall:**
Can get highest \( T_v \) displayed by \( N_r \) in \( O(|N_r| \cdot k^2) \).

**Definition**

\( r \) is stable on \( \iff \) all root- \( \) -paths contain \( r \)!

**Reduction Rule**

Root \( r \) of lowest tree-component \( N_r \) in \( N \) is stable on \( \iff \).  
\( \iff \) collapse \( N_r \) & the highest \( T_v \) with \( v \in T_v \) that \( N_r \) displays.

Choose the \( T_v \) containing the leaves that are accessible only via \( r \)!
Combining the Building Blocks

Recall:
Can get highest $T_v$ displayed by $N_r$ in $O(|N_r| \cdot k^2)$.

Definition
$r$ is stable on $r \iff$ all root- $r$ -paths contain $r$.

Reduction Rule
Root $r$ of lowest tree-component $N_r$ in $N$ is stable on $r$. 
$\Rightarrow$ collapse $N_r \&$ the highest $T_v$ with $r \in T_v$ that $N_r$ displays.

Choose the $T_v$ containing the leaves that are accessible only via $r$!

Theorem
Solve TC in $O(|N|)$ if (a) max. indegree (reti. chain) is constant & (b) all tree-component-roots are stable.

Corollary
Solve TC in $O(|N|)$ on (a) reticulation visible (all reti. stable) & (b) nearly-stable (unstable $\Rightarrow$ stable parent).

Corollary
TC can be solved in $O(|N|^{3})$ time on net-works $N$ in which all tree-vertices with reticulation parents are stable.
Combining the Building Blocks

**Network**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>r</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Tree**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>r</strong></td>
<td></td>
</tr>
</tbody>
</table>

**Recall:**

Can get highest $T_v$ displayed by $N_r$ in $O(|N_r| \cdot k^2)$.

**Definition**

$r$ is stable on $\iff$ all root-paths contain $r$.

**Reduction Rule**

Root $r$ of lowest tree-component $N_r$ in $N$ is stable on $\iff$ collapse $N_r$ & the highest $T_v$ with $r \in T_v$ that $N_r$ displays.

**Problem**

Which $T_v$ to choose?

Choose the $T_v$ containing the leaves that are accessible only via $r$!

**Theorem**

Solve TC in $O(|N|)$ if

1. max indegree (reti. chain) is constant $\neq$
2. all tree-component-roots are stable.

**Corollary**

Solve TC in $O(|N|)$ on

1. reticulation visible (all reti. stable)
2. nearly-stable (unstable $\Rightarrow$ stable parent).

**Corollary**

TC can be solved in $O(|N|^3)$ on networks $N$ in which all tree-vertices with reticulation parents are stable.
Combining the Building Blocks

Recall:
Can get highest $T_v$ displayed by $N_r$ in $O(|N_r| \cdot k^2)$

Definition
$r$ is stable on $\iff$ all root-paths contain $r$

Reduction Rule
Root $r$ of lowest tree-component $N_r$ in $N$ is stable on $\iff$
collapse $N_r$ & the highest $T_v$ with $r \in T_v$ that $N_r$ displays.

Choose the $T_v$ containing the leaves that are accessible only via $r$!

Theorem
solve TC in $O(|N|)$ if (a) max indegree (ret chain) is constant $\iff$
(b) all tree-component roots are stable.

Corollary
solve TC in $O(|N|)$ on (a) reticulation visible (all reti. stable) $\iff$
(b) nearly-stable (unstable $\Rightarrow$ stable parent)
Combining the Building Blocks

**Recall:**
Can get highest $T_v$ displayed by $N_r$ in $O(|N_r|)$.

**Theorem**
Solve $TC$ in $O(|N|)$ if (a) max indegree (reticulation chain) is constant \( \neq \) (b) all tree-component roots are stable.

**Corollary**
Solve $TC$ in $O(|N|)$ on (a) reticulation visible (all reti. stable) \( \neq \) (b) nearly-stable (unstable $\Rightarrow$ stable parent).

**Corollary**
$TC$ can be solved in $O(|N|^3)$ time on networks $N$ in which all tree-vertices with reticulation parents are stable.
General Networks

Problems

(a) Roots of tree-components may be unstable.
General Networks

Problems

(a) Roots of tree-components may be unstable.
General Networks

Problems

(a) Roots of tree-components may be unstable.
(b) Avoid using tree-components multiple times.

Network

Tree

Idea:

\[ [r, v] = 1 \quad \Leftrightarrow \quad N_r \text{ displays } T_v \]
General Networks

Problems
(a) Roots of tree-components may be unstable.
(b) Avoid using tree-components multiple times.

Network

Tree

Idea:
\[ [r, v, R] = 1 \iff N_r \text{ displays } T_v \text{ using unstable comp.-roots } R. \]
General Networks

Problems

(a) Roots of tree-components may be unstable.
(b) Avoid using tree-components multiple times.

Subdivision $S$ of $T_v$ is nice $\iff$
if $r \in S$, then $S$ contains all $l$ that $r$ is stable on.

Idea:

$[r, v, R] = 1$
$\iff$
$N_r$ displays $T_v$ using unstable comp.-roots $R$.

Theorem

solve TC in $O(3^t \cdot |N_r| \cdot |T_v|)$ time
$t = \#$ unstable comp.-roots.
General Networks

Problems

(a) Roots of tree-components may be unstable.
(b) Avoid using tree-components multiple times.

Subdivision $S$ of $T_v$ is nice $\iff$ if $r \in S$, then $S$ contains all $l$ that $r$ is stable on.

Sketch:

$S_1 \& S_2$ disjoint $\Rightarrow$ $S_1 \& S_2$ intersect $\Rightarrow$ share comp.-root $r$ stable $\Rightarrow$ shared leaf $E$ $r$ unstable $\Rightarrow$ $R_1 \& R_2$ intersect

Idea:

$[r, v, R] = 1$ $\iff$ $N_r$ displays $T_v$ using unstable comp.-roots $R$.

Theorem

Solve TC in $O(3^t \cdot |N| \cdot |T|)$ time $t = \#\text{unstable comp.-roots}$.
General Networks

Problems
(a) Roots of tree-components may be unstable.
(b) Avoid using tree-components multiple times.

Subdivision $S$ of $T_v$ is nice $\iff$
if $r \in S$, then $S$ contains all $l$ that $r$ is stable on.

Idea: $[r, v, R] = 1$ $\iff$ $N_r$ displays $T_v$ using unstable comp.-roots $R$.

Theorem solve TC in $O(3^t \cdot |N_r| \cdot |T_v|)$ time $t = \#$unstable comp.-roots.
General Networks

Problems

(a) Roots of tree-components may be unstable.
(b) Avoid using tree-components multiple times.

Subdivision $S$ of $T_v$ is nice $\iff$
if $r \in S$, then $S$ contains all $l$ that $r$ is stable on.

Idea:
\[ [r, v, R] = 1 \]
\[ \Leftrightarrow \]
$N_r$ displays $T_v$ using unstable comp.-roots $R$.

Theorem
solve TC in $O(3^t \cdot |N_r| \cdot |T_v|)$ time
$t = \#\text{unstable comp.-roots}$
General Networks

Problems

(a) Roots of tree-components may be unstable.

(b) Avoid using tree-components multiple times.

Subdivision $S$ of $T_v$ is nice $\iff$ if $r \in S$, then $S$ contains all $l$ that $r$ is stable on.

Idea:

$[r, v, R] = 1$ $\iff$ $N_r$ displays $T_v$ using unstable comp.-roots $R$.

Theorem

solve TC in $O(3^t \cdot |N_r| \cdot |T_v|)$ time

$t = \#\text{unstable comp.-roots}$
General Networks

Problems
(a) Roots of tree-components may be unstable.
(b) Avoid using tree-components multiple times.

Subdivision $S$ of $T_v$ is nice $\iff$
if $r \in S$, then $S$ contains all $l$ that $r$ is stable on.

Idea:
$[r, v, R] = 1$
$\iff$
$N_r$ displays $T_v$
using unstable comp.-roots $R$.

Sketch:
$S_1 \nmid S_2$ disjoint
$S_1 \nmid S_2$ have disjoint leaf-sets
$\Rightarrow S_1 \nmid S_2$ intersect
$\Rightarrow$ share comp.-root $r$
$r$ stable $\Rightarrow$ shared leaf $
$r$ unstable $\Rightarrow R_1 \nmid R_2$ intersect
General Networks

Problems
(a) Roots of tree-components may be unstable.
(b) Avoid using tree-components multiple times.

Subdivision $S$ of $T_v$ is nice $\iff$ if $r \in S$, then $S$ contains all $l$ that $r$ is stable on.

Sketch: $S_1 \nsubseteq S_2$ disjoint
$S_1 \nsubseteq S_2$ have disjoint leaf-sets
$\Rightarrow S_1 \nsubseteq S_2$ intersect
$\Rightarrow$ share comp.-root $r$

$r$ stable $\Rightarrow$ shared leaf $
=r$ unstable $\Rightarrow R_1 \nsubseteq R_2$ intersect $

Idea: $[r, v, R] = 1$
$\iff$
$N_r$ displays $T_v$
using unstable comp.-roots $R$. 

Theorem solve TC in $O(3^t \cdot |N| \cdot |T|)$ time
$t = \#\text{unstable comp.-roots}$
General Networks

Problems
(a) Roots of tree-components may be unstable.
(b) Avoid using tree-components multiple times.

Subdivision $S$ of $T_v$ is nice $\iff$ if $r \in S$, then $S$ contains all $l$ that $r$ is stable on.

Idea:
\[
[r, v, R] = 1 
\iff
N_r \text{ displays } T_v \text{ using unstable comp.-roots } R.
\]

Theorem
solve TC in \( O(3^t \cdot |N| \cdot |T|) \) time

\[ t = \#\text{unstable comp.-roots} \]

\text{S_2 disjoint}
\text{disjoint leaf-sets}
\text{intersect}
\text{comp.-root } r
\text{shared leaf } \]
\text{unstable } \Rightarrow R_1 \neq R_2 \text{ intersect}

36 / 38
Small Taxonomy of Network Classes

c.f. "Who is Who in Phylogenetic Networks" (http://phylnet.univ-mlv.fr/)
Thanks
& Enjoy
Part III