Algorithms in Bioinformatics

End of Clustering

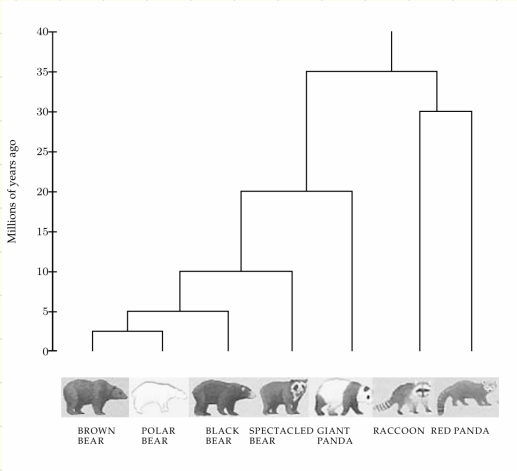
Recap

- HW due Tuesday
- Essay - up soon
- One last HW after (?) Thanksgiving (or 2?)
- Final writing assignment

Today: Evolutionary Trees (10.5)

Motivation: Common to use DNA similarity to study evolution patterns

Ex: [O'Brien et al 1985] used ~500,000 nucleotides to construct evolutionary tree of pandas & raccoon:



Many other such studies:

- mtDNA & "Out of Africa" claim

<u>Model</u>: rooted trees:

- internal vertices are (hypothetical) common ancestors
- leaves are existing species

Each root to leaf path is an evolutionary path.

(Some times unrooted are used, if no single common ancestor is assumed.)

Often, the weight of an edge uv, $w(uv)$, is # of mutations between u & v.

Can also have a length on each edge:

- $t(v)$ is the "moment" when species v produced descendents
- and $\ell(uv) = w(u) - w(v)$

Tree reconstruction problem:

- Given an $n \times n$ distance matrix $(D_{i,j})$, can we build a tree with n leaves such that $d_T(i,j) = D_{i,j}$ for each pair of leaves i,j?

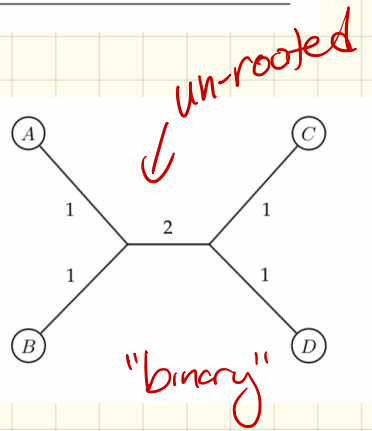
Distance-Based Phylogeny Problem:
Reconstruct an evolutionary tree from a distance matrix.

Input: An $n \times n$ distance matrix $(D_{i,j})$.

Output: A weighted unrooted tree T with n leaves fitting D, that is, a tree such that $d_{i,j}(T) = D_{i,j}$ for all $1 \leq i < j \leq n$ if $(D_{i,j})$ is additive.

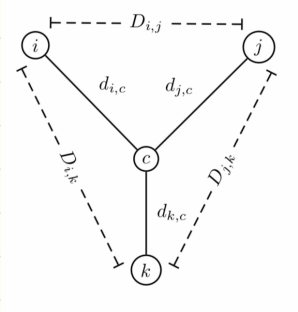
Ex:

	A	B	C	D
A	0	2	4	4
B	2	0	4	4
C	4	4	0	2
D	4	4	2	0



called <u>additive</u> if $\exists$ a tree

Can always do this for 3×3 matrix:



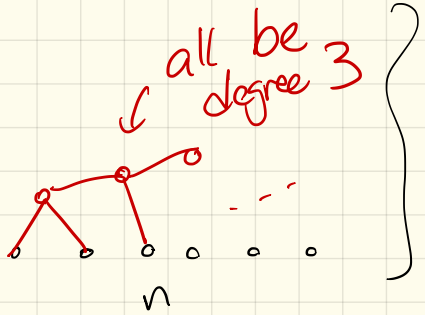
$$d_{i,c} + d_{j,c} = D_{i,j} \qquad d_{i,c} + d_{k,c} = D_{i,k} \qquad d_{j,c} + d_{k,c} = D_{j,k}.$$

The solution is given by

$$d_{i,c} = \frac{D_{i,j} + D_{i,k} - D_{j,k}}{2} d_{j,c} = \frac{D_{j,i} + D_{j,k} - D_{i,k}}{2} d_{k,c} = \frac{D_{k,i} + D_{k,j} - D_{i,j}}{2}.$$

For an $n \times n$ matrix:

- need n leaves



so $n-2$ internal vertices (if "unrooted" binary tree)

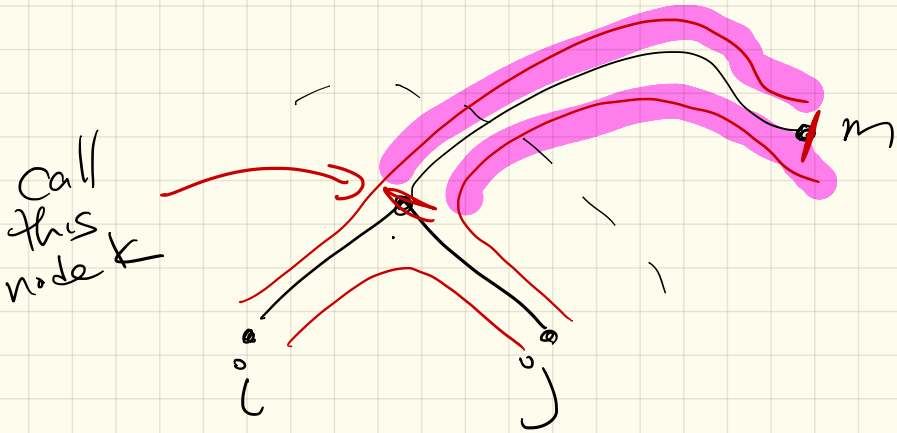
$\Rightarrow$ and $2n-3$ edges

Result: $\binom{n}{2}$ equations & $2n-3$ variables

Won't always have a solution, but if there is one, we can find it...

Goal: Be just a little greedy.

Find a pair of neighboring leaves:



For all other leaves m,

$$D_{k,m} = \frac{D_{i,m} + D_{j,m} - D_{i,j}}{2}$$

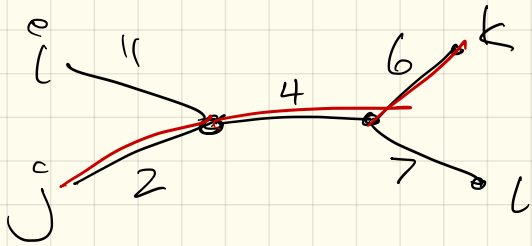
So: If you can find i & j, can remove them from the matrix & replace with k.

Just put $D_{k,m} = \frac{D_{i,m} + D_{j,m} - D_{i,j}}{2}$

in every (k,m) slot in the matrix

How hard is it to find neighboring leaves?

- Can't choose closest i & j



(j & k are not nbrs)

Instead: Imagine shortening all the leaves by δ!

	A	B	C	D
A	0	4	10	9
B	4	0	8	7
C	10	8	0	9
D	9	7	9	0

$\delta = 1$

	A	B	C	D
A	0	2	8	7
B	2	0	6	5
C	8	6	0	7
D	7	5	7	0

$i \leftarrow A$
$j \leftarrow B$
$k \leftarrow C$

	A	C	D
A	0	8	7
C	8	0	7
D	7	7	0

$\delta = 3$

	A	C	D
A	0	2	1
C	2	0	1
D	1	1	0

$i \leftarrow A$
$j \leftarrow D$
$k \leftarrow C$

	A	C
A	0	2
C	2	0

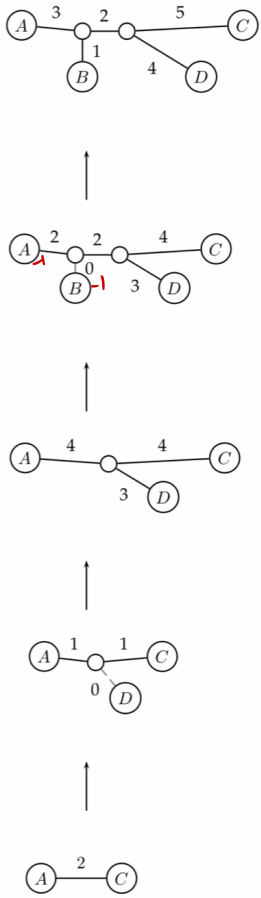


Figure 10.14 The iterative process of shortening the hanging edges of a tree.

Finding the "right" δ:

- not explicitly recorded, but <u>is</u> in there implicitly

<u>Dfn</u>: A triple i, j, k is <u>degenerate</u> if

$$D_{i,j} + D_{j,k} = D_{i,k}$$

Meaning:

If there is a degenerate triple, we call the entire matrix D degenerate.

Cool trick: just remove j!

$\Rightarrow$ Get $(n-1)\times(n-1)$ matrix, & readd j after i + j's path in T is determined.

Now, if <u>not</u> degenerate:

- Shorten all leaves by δ until one forms.

ie the minimum δ s.t. $D_{i,j} - 2\delta$ has a degenerate triple.

(δ is called the "trimming parameter")

This gives an algorithm:

```
ADDITIVEPHYLOGENY(D)
if D is a 2 × 2 matrix
    T ← the tree consisting of a single edge of length D_{1,2}.
    return T
if D is non-degenerate
    δ ← trimming parameter of matrix D
    for all 1 ≤ i ≠ j ≤ n
        D_{i,j} ← D_{i,j} − 2δ
else
    δ ← 0
Find a triple i, j, k in D such that D_{ij} + D_{jk} = D_{ik}
x ← D_{i,j}
Remove jth row and jth column from D.
T ← ADDITIVEPHYLOGENY(D)
Add a new vertex v to T at distance x from i to k
Add j back to T by creating an edge (v, j) of length 0
for every leaf l in T
    if distance from l to v in the tree T does not equal D_{l,j}
        output "Matrix D is not additive"
        return
Extend hanging edges leading to all leaves by δ
return T
```

HW

A faster approach:

"Four point condition"

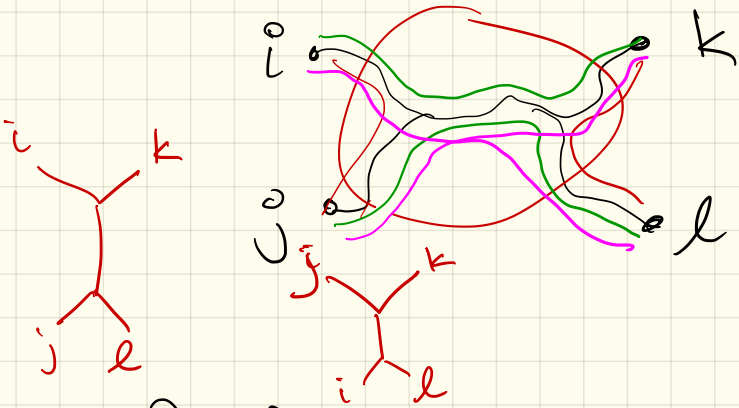
Consider 4 indices $1 \le i, j, k, l \le n$ (all distinct).

Compute: $\underline{D_{i,j} + D_{k,l}}$

$\underline{D_{i,k} + D_{j,l}}$

$\underline{D_{i,l} + D_{j,k}}$

If additive: Tree



2 of these are equal & the 3rd must be smaller.

<u>Thm</u>: A matrix is additive $\iff$ 4 point condition holds for every 4 distinct elements.

(I won't prove today — one way is obvious, & the other much less so.)

If not additive, one natural question is, can we get <u>close</u> to D in a tree?

Least Squares Distance-Based Phylogeny Problem:
Given a distance matrix, find the evolutionary tree that minimizes squared error.

Input: An $n \times n$ distance matrix $(D_{i,j})$

Output: A weighted tree T with n leaves minimizing $\sum_{i,j}(d_{i,j}(T) - D_{i,j})^2$ over all weighted trees with n leaves.

This is NP-Hard.

find an approximate tree

Using hierarchical clustering:

<u>UPGMA</u>: Unweighted Pair Group Method w/ Arithmetic mean

- Goal: assign heights to vertices of your tree

  length of u, v is difference in heights

- Given clusters C_1 + C_2

$$D(C_1, C_2) = \frac{1}{|C_1||C_2|} \sum_{i \in C_1} \sum_{j \in C_2} D(i,j)$$

(ie average pairwise distance)

The algorithm choose the 2 closet clusters to merge

```
UPGMA(D, n)
Form n clusters, each with a single element
Construct a graph T by assigning an isolated vertex to each cluster
Assign height h(v) = 0 to every vertex v in this graph
while there is more than one cluster
    Find the two closest clusters C_1 and C_2
    Merge C_1 and C_2 into a new cluster C with |C_1| + |C_2| elements
    for every cluster C* ≠ C
        D(C, C*) = 1/(|C|·|C*|) Σ_{i∈C} Σ_{j∈C*} D(i, j)
    Add a new vertex C to T and connect to vertices C_1 and C_2
    h(C) ← D(C_1, C_2)/2
    Assign length h(C) − h(C_1) to the edge (C_1, C)
    Assign length h(C) − h(C_2) to the edge (C_2, C)
    Remove rows and columns of D corresponding to C_1 and C_2
    Add a row and column to D for the new cluster C
return T
```

This actually produces an ultrametric (where distance to the root from any leaf is identical.)

Improvement [Saitou-Nei 1987]:

- Bring back earlier neighbor joining idea.

Incorporate separation:

$$u(C) = \frac{1}{\#\text{clusters} - 2} \sum_{\substack{\text{all} \\ \text{clusters } C'}} D(C, C')$$

"mysterious"

Choose 2 nearby clusters (as in last alg) that are also far from others

goal: minimize $D(C_1, C_2)$
& maximize $u(C_1) + u(C_2)$

(In reality, minimize

$$D(C_1, C_2) - u(C_1) - u(C_2)$$

Algorithm:

NEIGHBORJOINING(D, n)

Form n clusters, each with a single element
Construct a graph T by assigning an isolated vertex to each cluster
while there is more than one cluster
    Find clusters C_1 and C_2 minimizing $D(C_1, C_2) - u(C_1) - u(C_2)$
    Merge C_1 and C_2 into a new cluster C with $|C_1| + |C_2|$ elements
    Compute $D(C, C^*) = \frac{D(C_1,C)+D(C_2,C)}{2}$ to every other cluster C^*
    Add a new vertex C to T and connect it to vertices C_1 and C_2
    Assign length $\frac{1}{2}D(C_1, C_2) + \frac{1}{2}(u(C_1) - u(C_2))$ to the edge (C_1, C)
    Assign length $\frac{1}{2}D(C_1, C_2) + \frac{1}{2}(u(C_2) - u(C_1))$ to the edge (C_2, C)
    Remove rows and columns of D corresponding to C_1 and C_2
    Add a row and column to D for the new cluster C
return T

Works well in practice:

- doesn't assume ideal "clock" measuring distance to the root)
- neighbors are "close", but also "far" from rest of the tree.

Another method: (10.9)

Scrap the distance matrix approach entirely.

Instead, use alignment matrix:

n species, each with m nucleotides

so $n \times m$ matrix

↳ ~~also~~ the characters

Goal: Construct a tree with n species at the leaves, where the internal vertices correspond to ancestral ones

Note: "character" is misleading (Might be # of legs, or any species attribute)

"Parsimony": minimize the total # of mutations.

Given a tree T where every vertex gets an m-long string,

$$d(\text{edge } uv) = d_H(v,w)$$

$$\text{Parsimony}(T) = \sum_{\substack{\text{all edges} \\ u,v \in T}} d_H(uv)$$

Of course, internal strings are initally unknown.

So goal is to find the strings and the tree structure to minimize the score.

small parsimony

large parsimony

So 2 versions...

#D

Small Parsimony Problem:
Find the most parsimonious labeling of the internal vertices in an evolutionary tree.

Input: Tree T with each leaf labeled by an m-character string. ↑single character

Output: Labeling of internal vertices of the tree T minimizing the parsimony score.

Note: can actually solve independently for each character.

First, we'll look at a version that introduces a scoring matrix:

Weighted Small Parsimony Problem:
Find the minimal weighted parsimony score labeling of the internal vertices in an evolutionary tree.

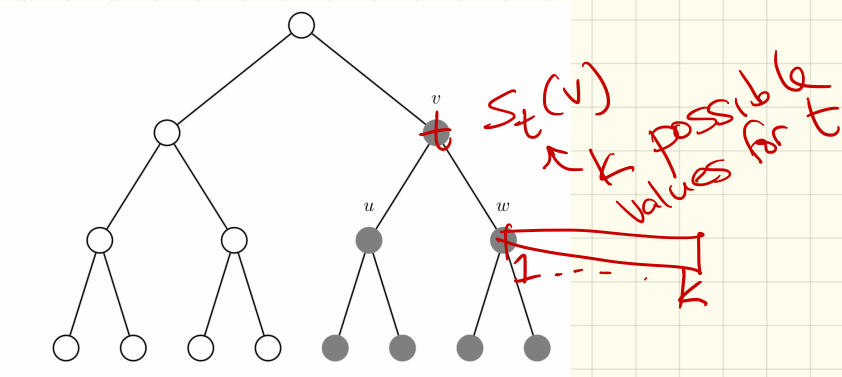
Input: Tree T with each leaf labeled by elements of a k-letter alphabet and a $k \times k$ scoring matrix (δ_{ij}).

Output: Labeling of internal vertices of the tree T minimizing the weighted parsimony score.

Instead of $d_H(v,w) = 0$ or 1, instead allow arbitrary scores $\delta_{i,j}$.

Solution: Dynamic Programming [Sankoff 1975]

Let $S_t(v)$ = minimum parsimony score of subtree rooted at v assuming it has character t.



then $S_t(v)$ can be computed using $S_1(u), S_2(u), \ldots, S_k(u)$ and $S_1(w), S_2(w), \ldots S_k(w)$

$$S_t(u) = \min_i \{S_i(u) + \delta_{i,t}\} + \min_j \{S_j(w) + \delta_{j,t}\}$$

for all k possibilities

At the leaves:

$s_t(v) = 0$ if v has letter t

$s_t(v) = \infty$ otherwise

δ	A	T	G	C
A	0	3	4	9
T	3	0	2	4
G	4	2	0	4
C	9	4	4	0

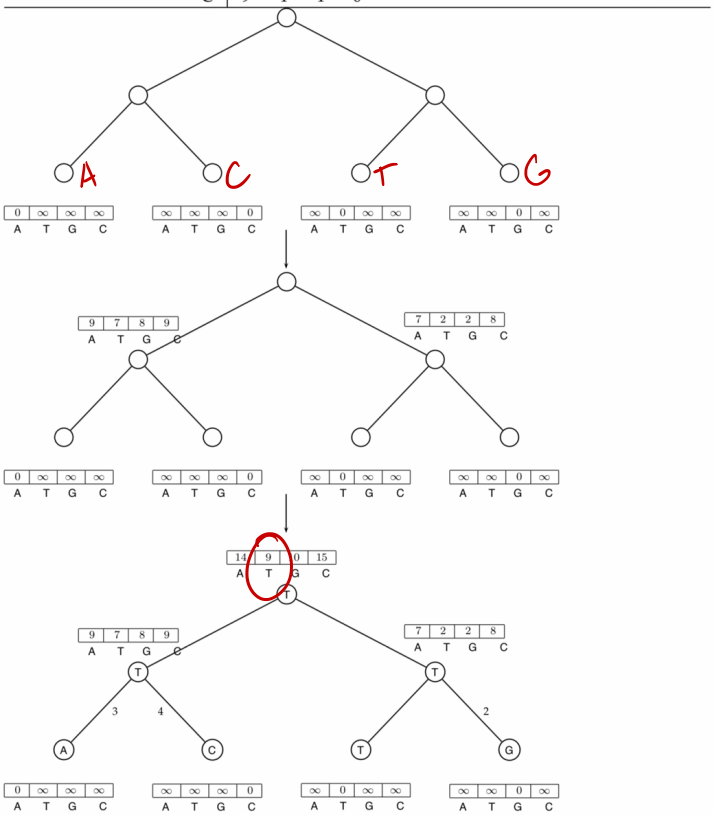


Figure 10.18 An illustration of Sankoff's algorithm. The leaves of the tree are labeled by A, C, T, G in order. The minimum weighted parsimony score is given by $s_T(root) = 0 + 0 + 3 + 4 + 0 + 2 = 9$.

What if we aren't given the tree structure?

Large Parsimony Problem:
Find a tree with n leaves having the minimal parsimony score.

Input: An $n \times m$ matrix M describing n species, each represented by an m-character string.

Output: A tree T with n leaves labeled by the n rows of matrix M, and a labeling of the internal vertices of this tree such that the parsimony score is minimized over all possible trees and over all possible labelings of internal vertices.

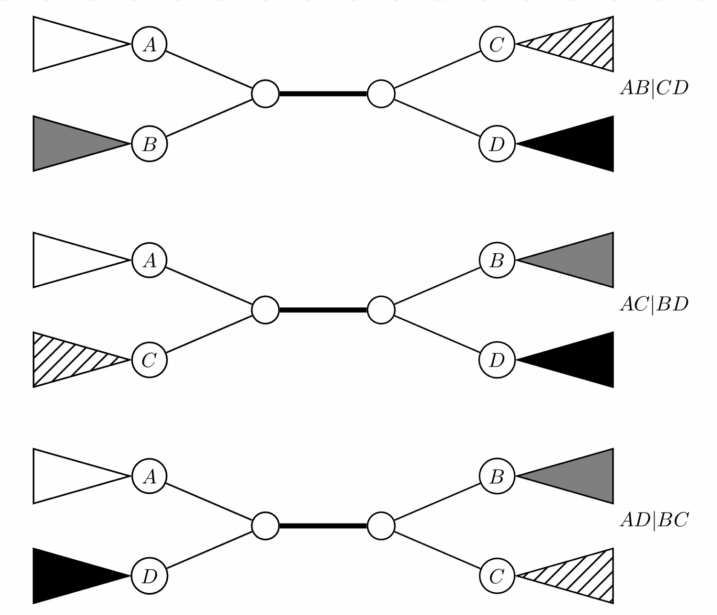
This one is NP-Complete.

Can brute force all trees + solve each small parsimony problem, but there are an exponential # of possible trees.

So heuristics are used.

Nearest Neighbor interchange:

Every internal edge defines four subtrees.

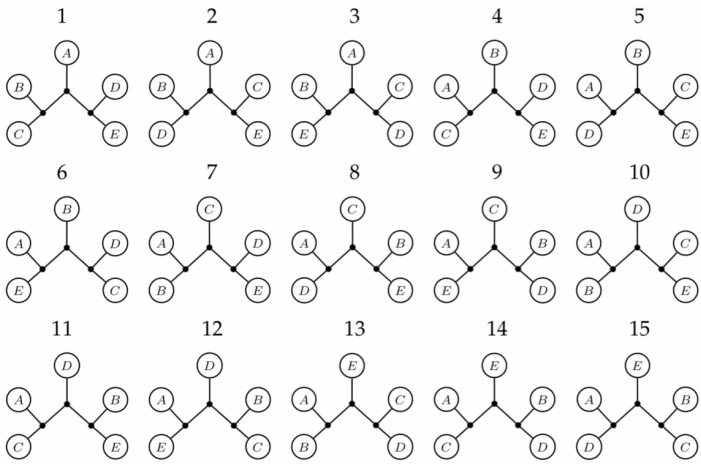


Call these <u>neighbor</u> configurations.

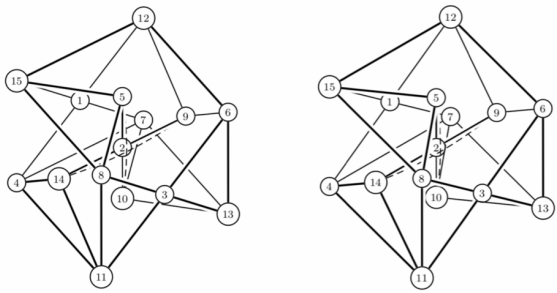
Greedy approach:

- Start w/ an arbitrary tree.
- Move to a neighbor if the score increases.

Picture:



(a) All 5-leaf binary trees



(b) Stereo projection of graph of trees

Downside: - no known approx guarantee (??)

- won't give OPT