Lecture 21: Imperfect Tree Construction

Study Chapter 10.9 – 10.11

Least-Squares Distance Phylogeny Problem

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

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

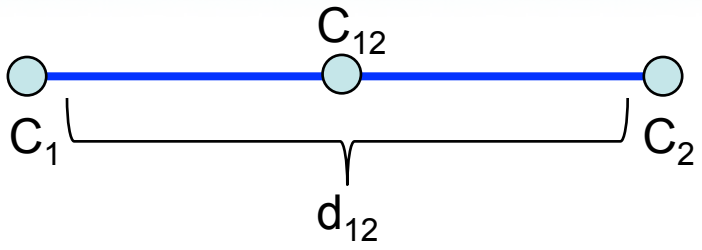
- Squared Error is a measure of the quality of the fit between distance matrix and the tree: we want to minimize it.

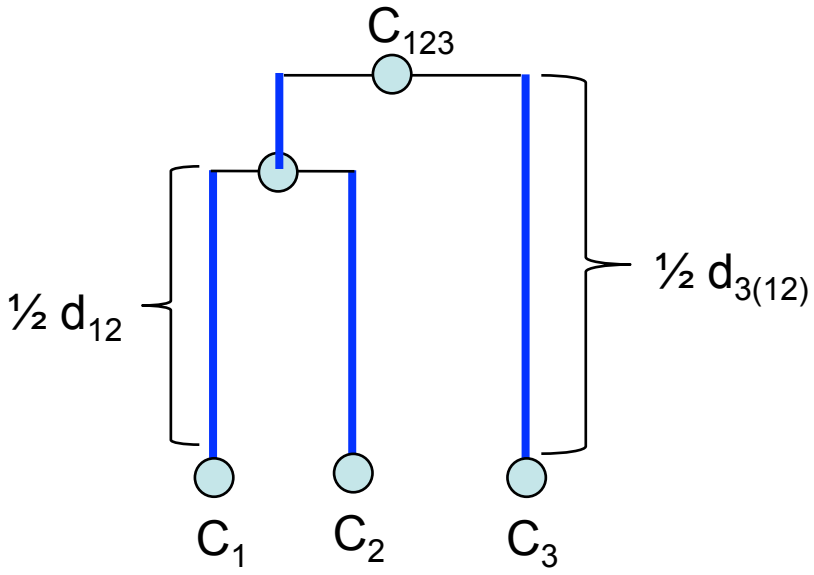
- **Least Squares Distance Phylogeny Problem**: finding the best approximation tree T for a non-additive matrix D (NP-hard).

UPGMA

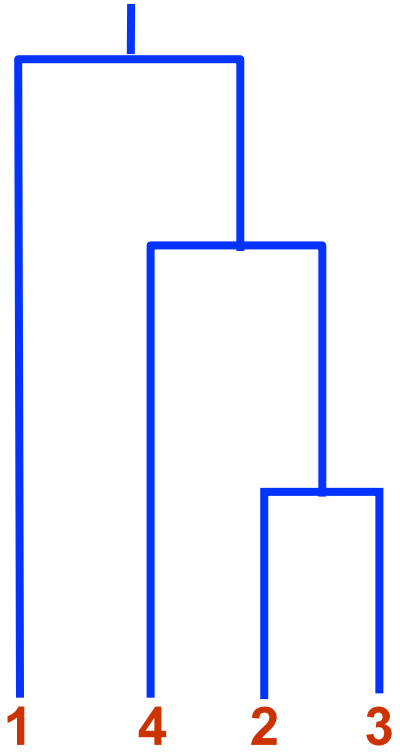
- Unweighted Pair Group Method with Arithmetic Mean (UPGMA)
- UPGMA is a hierarchical clustering algorithm:
  - assigns the distance between clusters to be the average pairwise distance
  - assigns a *height* to every vertex in the tree, that is midway between the cluster distances

"Balanced" Cluster Merging

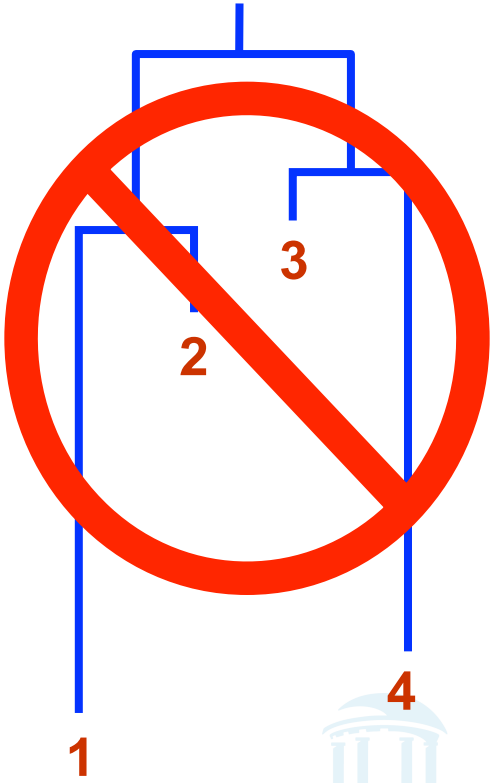




UPGMA generates trees like this



But never trees like this



UPGMA's Weakness

- The algorithm produces an *ultrametric* tree : the distance from the root to every leaf is the same
- UPGMA models a constant molecular clock:
  - all species represented by the leaves in the tree
  - assumed to coexist at t=0 and to have accumulated mutations (and thus evolve) at the same rate.
- In reality the assumptions of UPGMA are seldom true, but they are frequently approximately true.

Clustering in UPGMA

Given two disjoint clusters C_i, C_j of sequences,

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

Note that if $C_k = C_i \cup C_j$, then the distance to another cluster C_l is:

$$d_{kl} = \frac{d_{il}|C_i| + d_{jl}|C_j|}{|C_i| + |C_j|}$$

UPGMA Algorithm

Initialization:

Assign each x_i to its own cluster C_i

Define one leaf per sequence, each at height 0

Iteration:

Find two clusters C_i *and* C_j such that d_{ij} is min

Let $C_k = C_i \cup C_j$

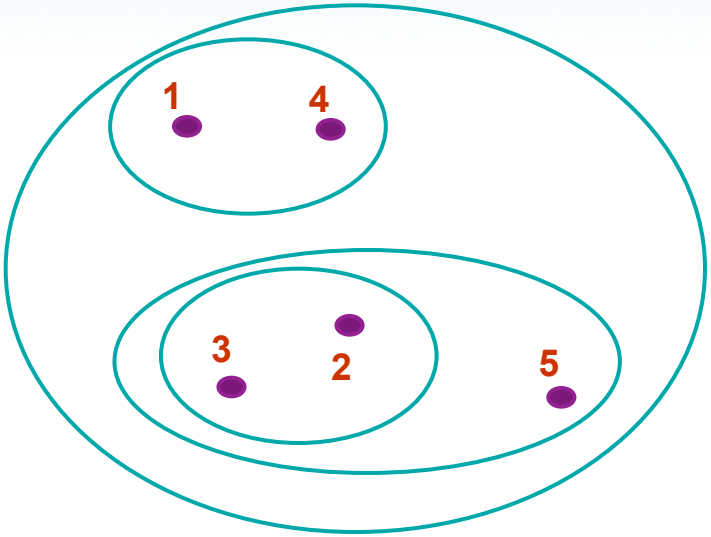
Add a vertex connecting C_i, C_j *and* place it at height $d_{ij}/2$

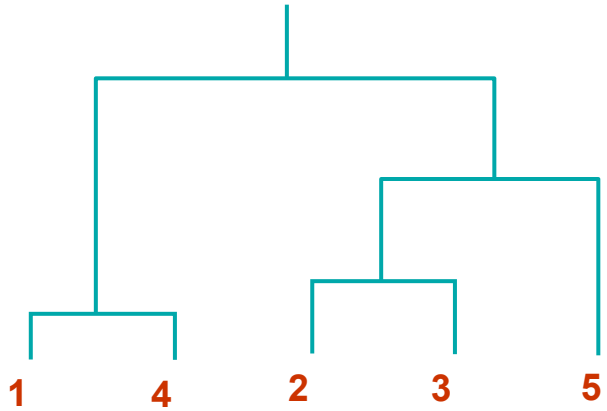
Delete C_i *and* C_j

Termination:

When a single cluster remains

UPGMA Algorithm (cont'd)

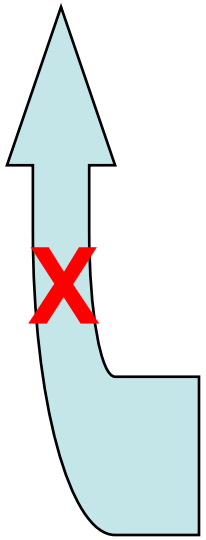




Alignment Matrix vs. Distance Matrix

Sequence a gene of length *m* nucleotides in *n* species to generate an…

n x *m* alignment matrix



CANNOT be transformed back into alignment matrix because information was lost on the forward transformation

Transform into…

n x *n* distance matrix

Character-Based Tree Reconstruction

- **Better technique**:
  - Character-based reconstruction algorithms use the n x m alignment matrix

    (n = # species, m = #characters)

    directly instead of using distance matrix.
  - **GOAL**: determine what character strings at internal nodes would best explain the character strings for the n observed species

Character-Based Tree Reconstruction

- Characters may be nucleotides of an aligned DNA, where A, G, C, T, - are *states* of this character
- Other characters may be the # of eyes or legs or the shape of a beak or a fin.
- By setting the length of an edge in the tree to the Hamming distance, we may define the **parsimony score** of the tree as the sum of the lengths (weights) of the edges

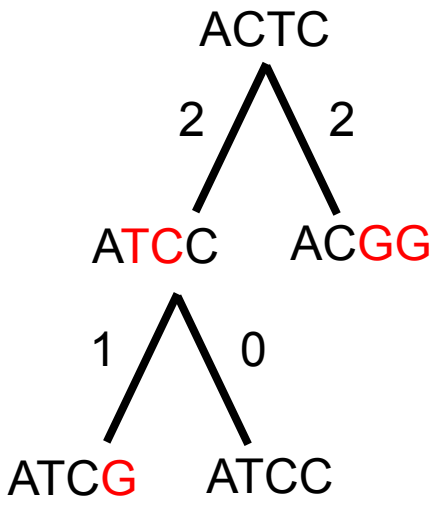
Parsimony Approach to Evolutionary Tree Reconstruction

- Assumes observed character differences result from the simplest possible, most parsimonious, explanation (i.e. the fewest mutations)
- Seeks the tree that yields lowest possible *parsimony score* - sum of cost of all changes mutations found in the tree
- Example: What is the most parsimonious ancestor to the following three sequences:

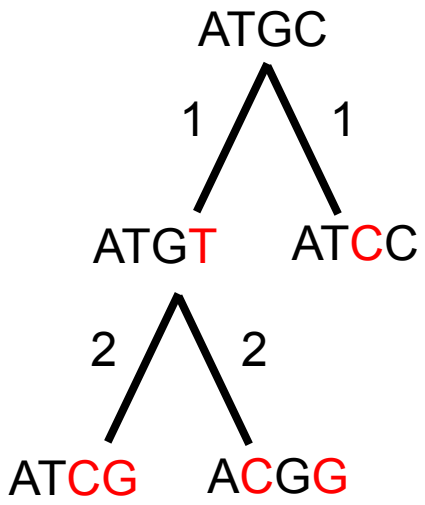
{ATCG, ATCC, ACGG}

Parsimony Scores

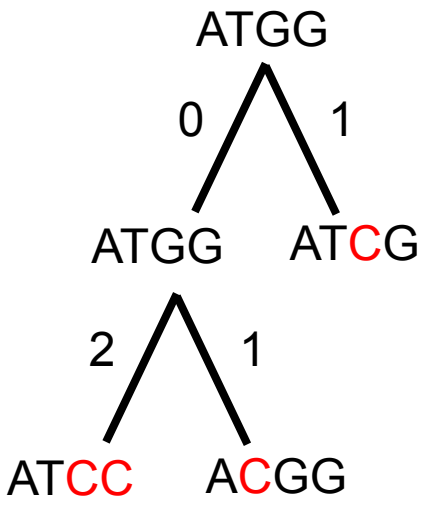
- Given ancestors and a tree relating them to the leafs, it is a simple matter to compute a parsimony score



Parsimony score: 5

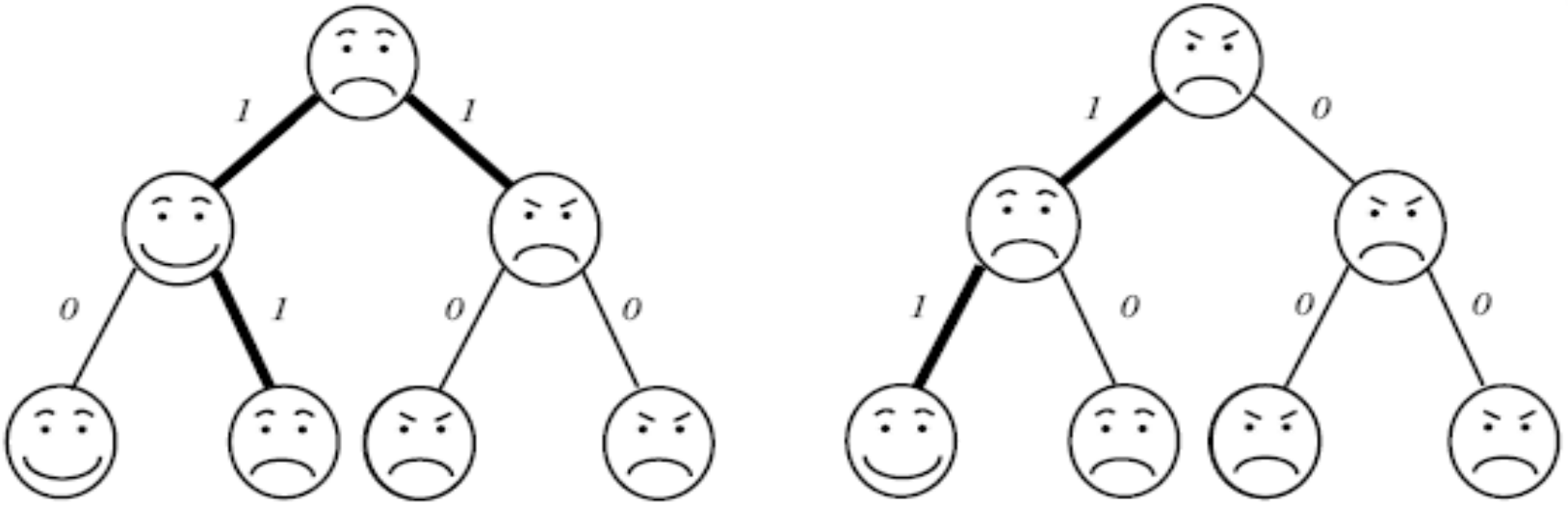


Parsimony score: 6



Parsimony score: 4

Character-Based Tree Reconstruction



(a) *Parsimony Score=3* (b) *Parsimony Score=2*

Figure 10.16 If we label a tree's leaves with characters (in this case, eyebrows and mouth, each with two states), and choose labels for each internal vertex, we implicitly create a *parsimony* score for the tree. By changing the labels in (a) we are able to create a tree with a better parsimony score in (b).

Small Parsimony Problem

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

- Output: Labeling of internal vertices of the tree T with ancestors that minimize the parsimony score.

- We can assume that every leaf is labeled by a single character, because the characters in the string are *independent*.

Weighted Small Parsimony Problem

- A more general version of Small Parsimony Problem
- Input includes a $k \times k$ scoring matrix describing the cost of transformation of each of k states into another one
- For the Small Parsimony problem, the scoring matrix is simply the Hamming distance

  $d_H(v, w) = 0$ if $v=w$

  $d_H(v, w) = 1$ otherwise

Scoring Matrices

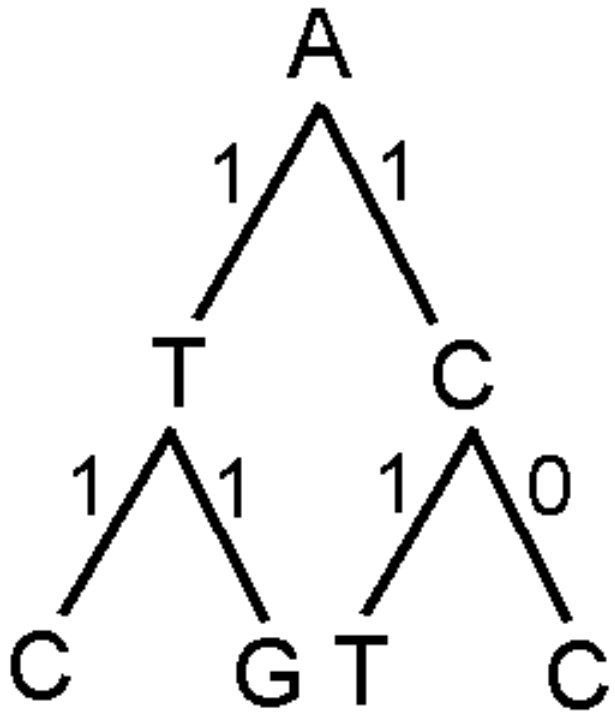
Small Parsimony Problem

	A	T	G	C
A	0	1	1	1
T	1	0	1	1
G	1	1	0	1
C	1	1	1	0

Weighted Parsimony Problem

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

Unweighted vs. Weighted

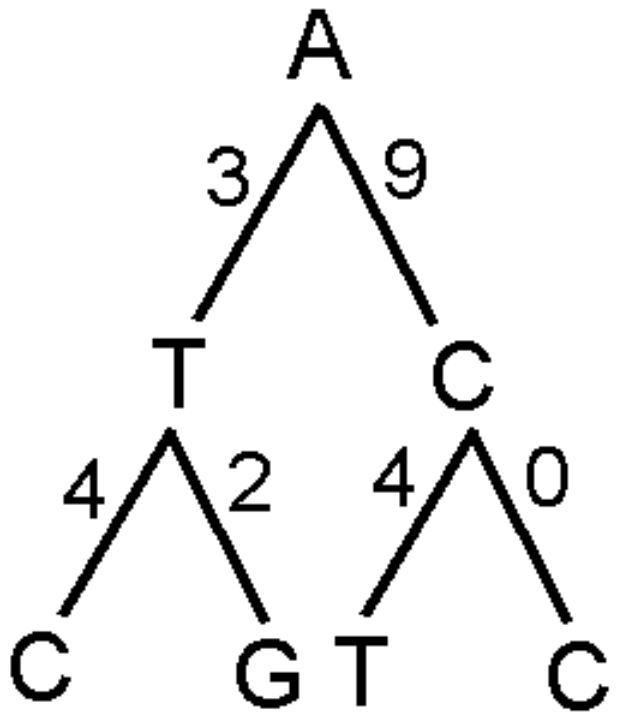


Small Parsimony Scoring Matrix:

	A	T	G	C
A	0	1	1	1
T	1	0	1	1
G	1	1	0	1
C	1	1	1	0

Small Parsimony Score: 5

Unweighted vs. Weighted



Weighted Parsimony Scoring Matrix:

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

Weighted Parsimony Score: 22

Weighted Small Parsimony Problem: Formulation

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

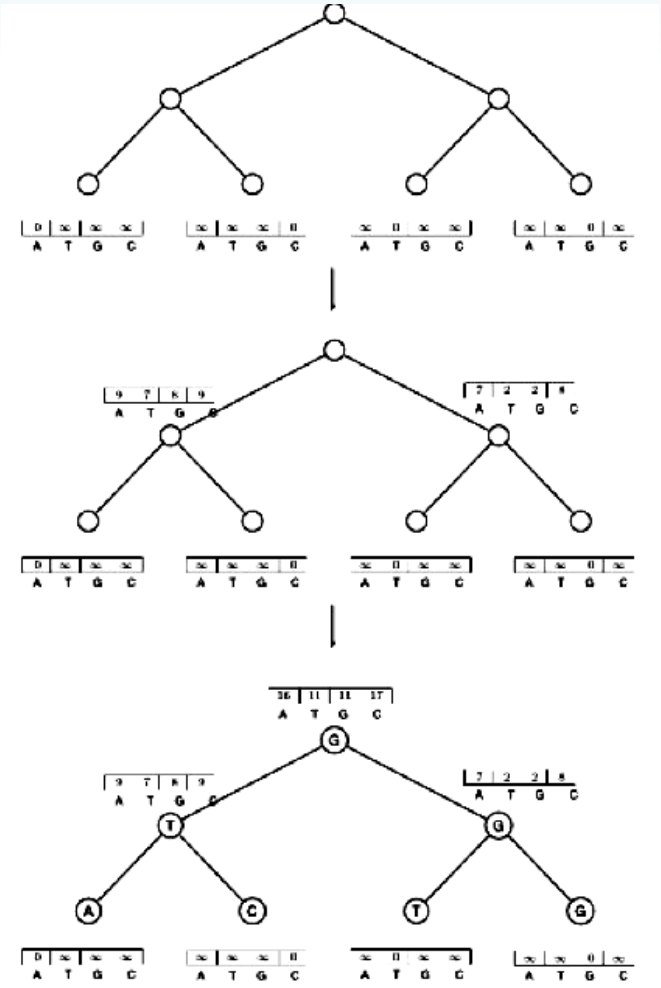
- Output: Labeling of internal vertices of the tree T minimizing the weighted parsimony score

Sankoff's Algorithm

- Check the children of a vertex and determine the minimum between them

- An example

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

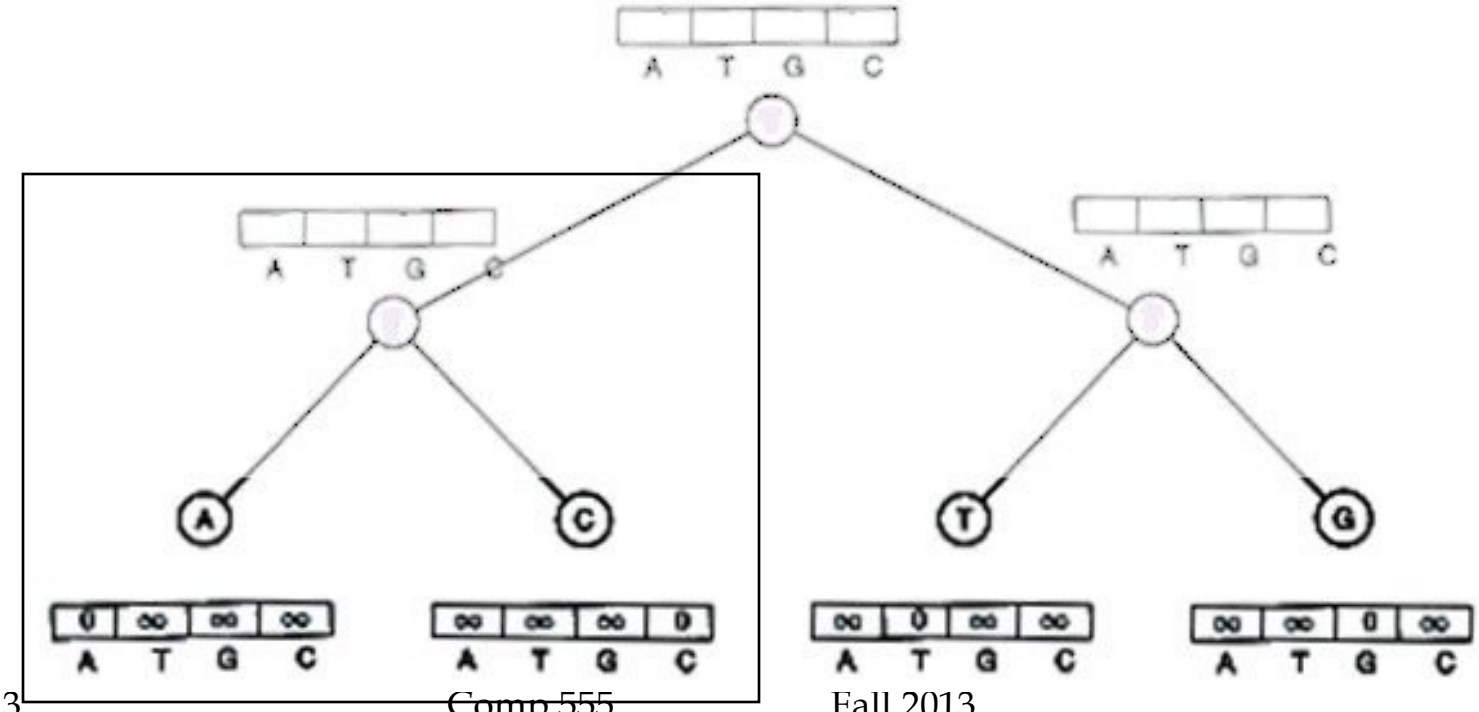


Sankoff Algorithm: Dynamic Programming

- Calculate and keep track of a score for every possible label at each vertex
  - $s_t(v)$ = minimum parsimony score of the **subtree** rooted at vertex v if v has character t
- The score at each vertex is based on scores of its children:
  - $s_t(\textbf{\textit{parent}}) = \min_i \{s_i(\textbf{\textit{ left child }}) + \delta_{i,t}\} + \min_j \{s_j(\textbf{\textit{ right child }}) + \delta_{j,t}\}$

Sankoff Algorithm

- Begin at leaves:
  - If leaf has the character in question, score is 0
  - Else, score is ∞

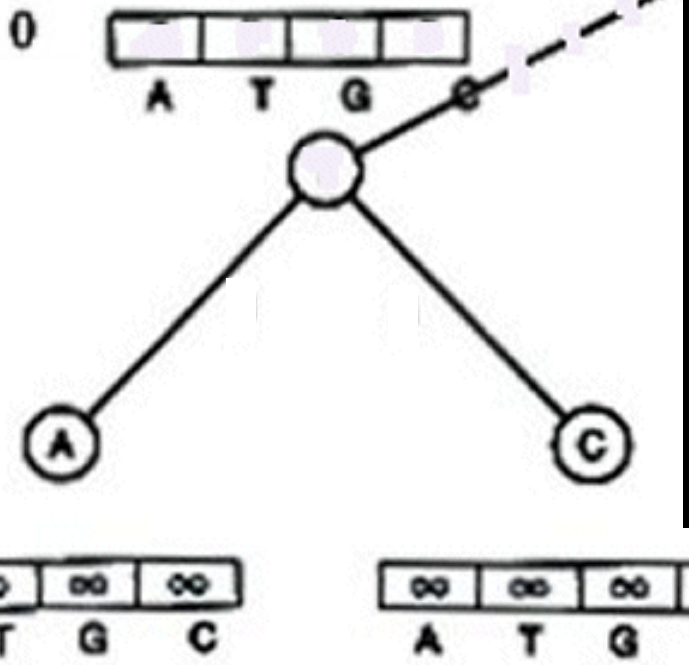


Sankoff Algorithm

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

$$s_t(v) = \min_i \{s_i(u) + \delta_{i,\,t}\} + \min_j \{s_j(w) + \delta_{j,\,t}\}$$

$$s_A(v) = 0 + \min_j \{s_j(w) + \delta_{j,\,A}\}$$



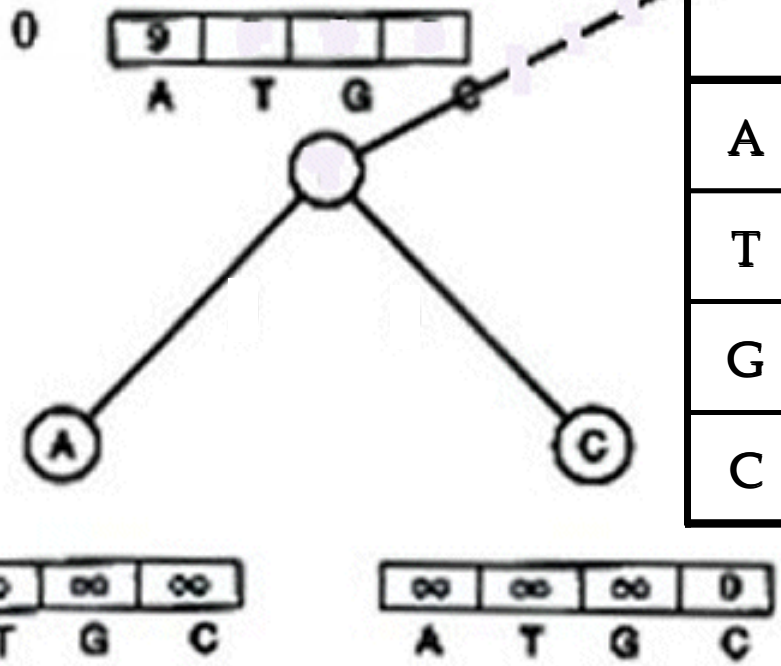
	$s_i(u)$	$\delta_{i,\,A}$	sum
A	0	0	**0**
T	∞	3	∞
G	∞	4	∞
C	∞	9	∞

Sankoff Algorithm (cont.)

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

$$s_t(v) = \min_i \{s_i(u) + \delta_{i,t}\} + \min_j \{s_j(w) + \delta_{j,t}\}$$

$s_A(v) = 0$
$+ 9 = \mathbf{9}$



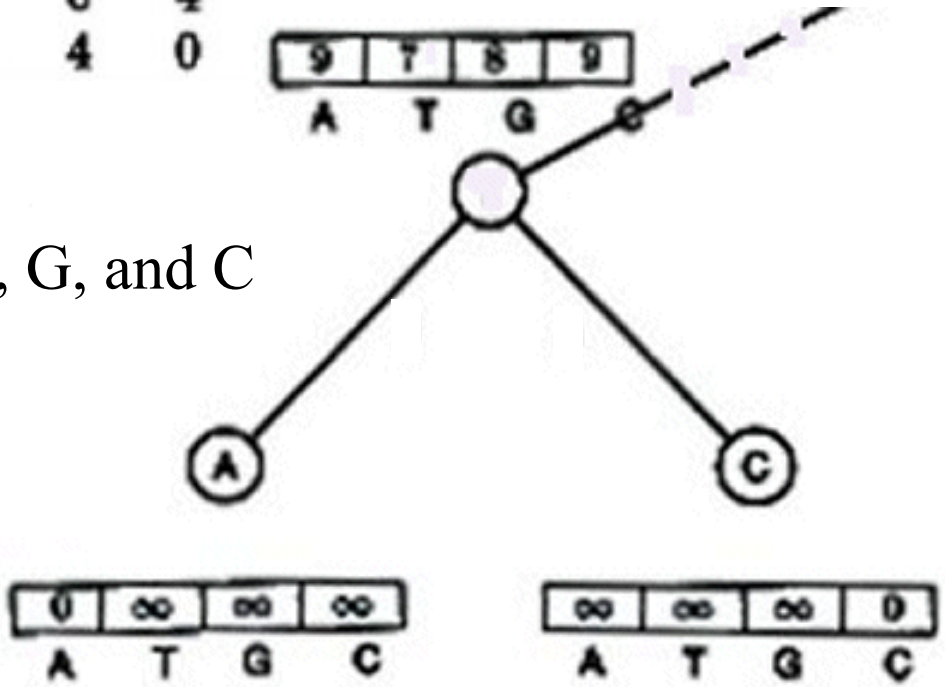
	$s_j(w)$	$\delta_{j,A}$	sum
A	∞	0	∞
T	∞	3	∞
G	∞	4	∞
C	0	9	**9**

Sankoff Algorithm (cont.)

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

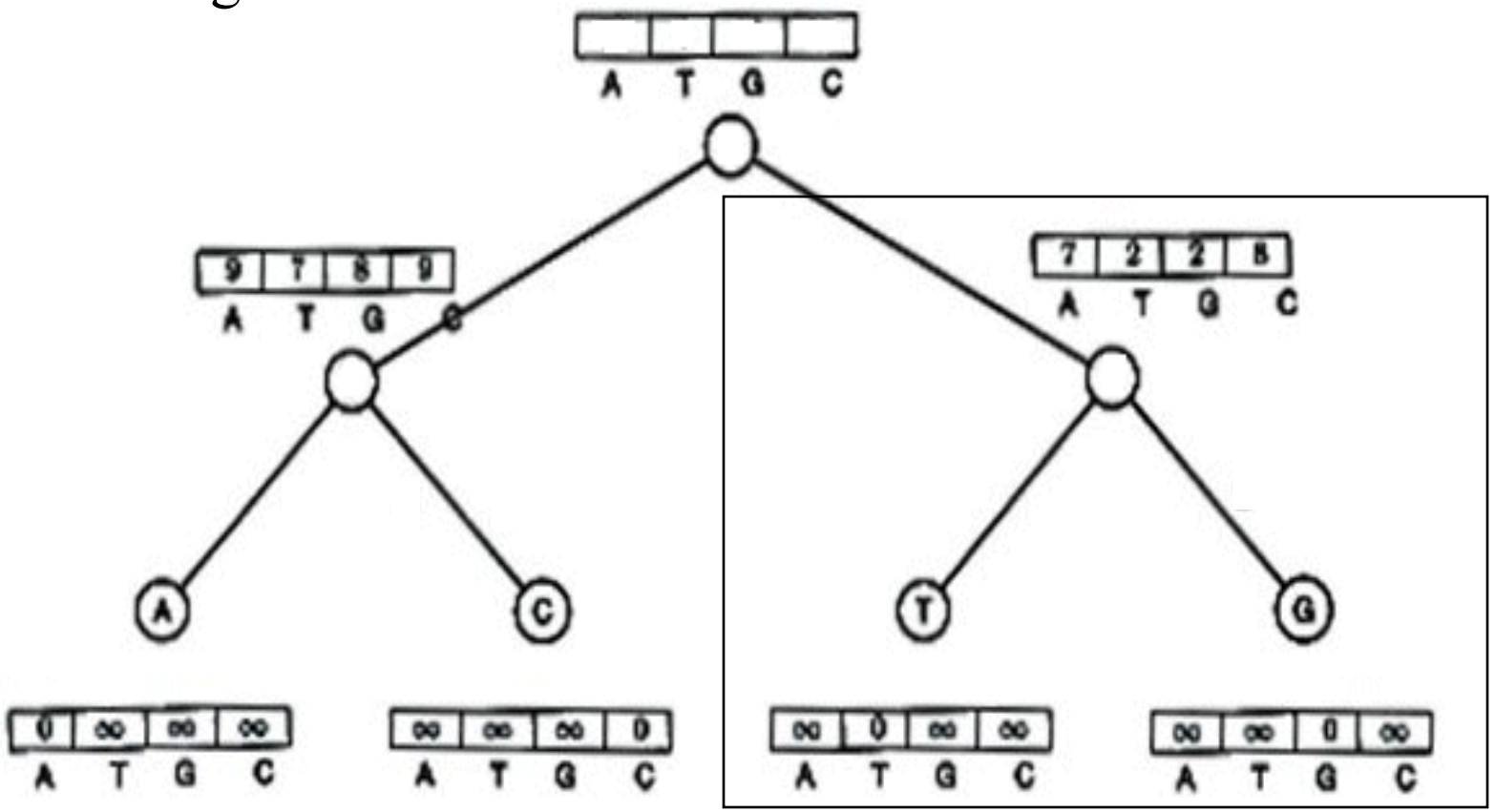
$$s_t(v) = \min_i \{s_i(u) + \delta_{i,t}\} + \min_j \{s_j(w) + \delta_{j,t}\}$$

Repeat for T, G, and C



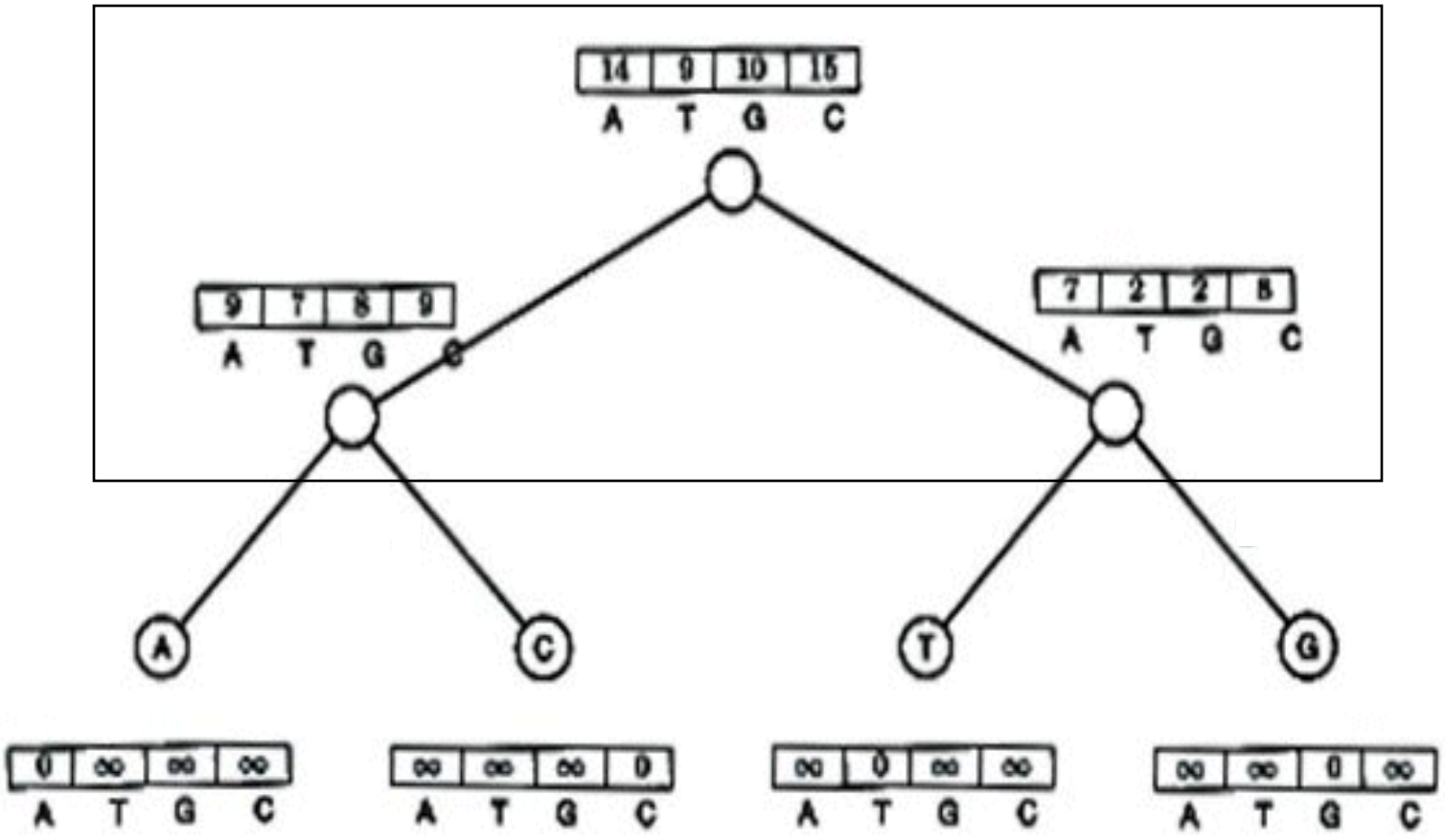
Sankoff Algorithm (cont.)

Repeat for right subtree



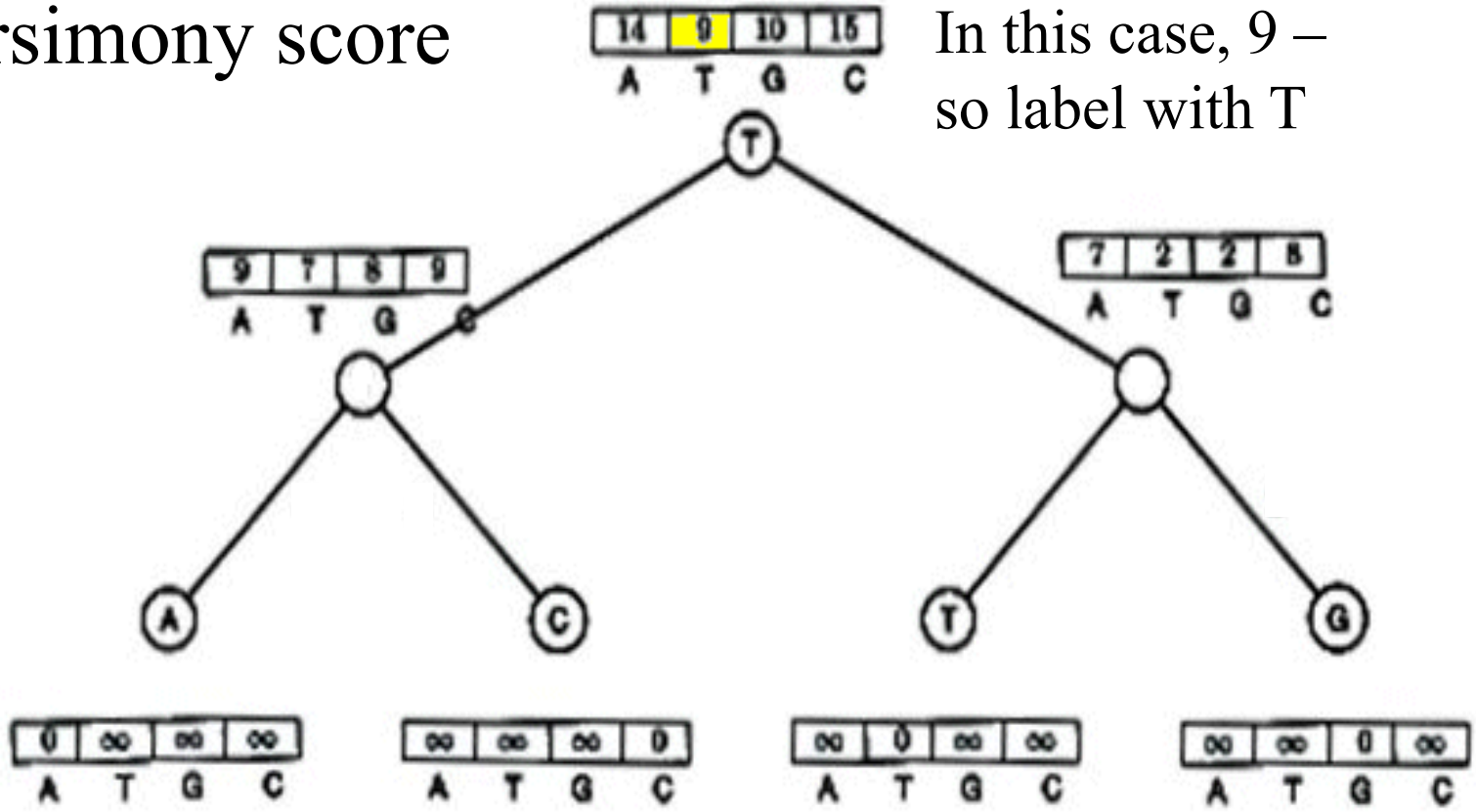
Sankoff Algorithm (cont.)

Repeat for root



Sankoff Algorithm (cont.)

Smallest score at root is minimum weighted parsimony score



In this case, 9 – so label with T

Sankoff Algorithm: Traveling down the Tree

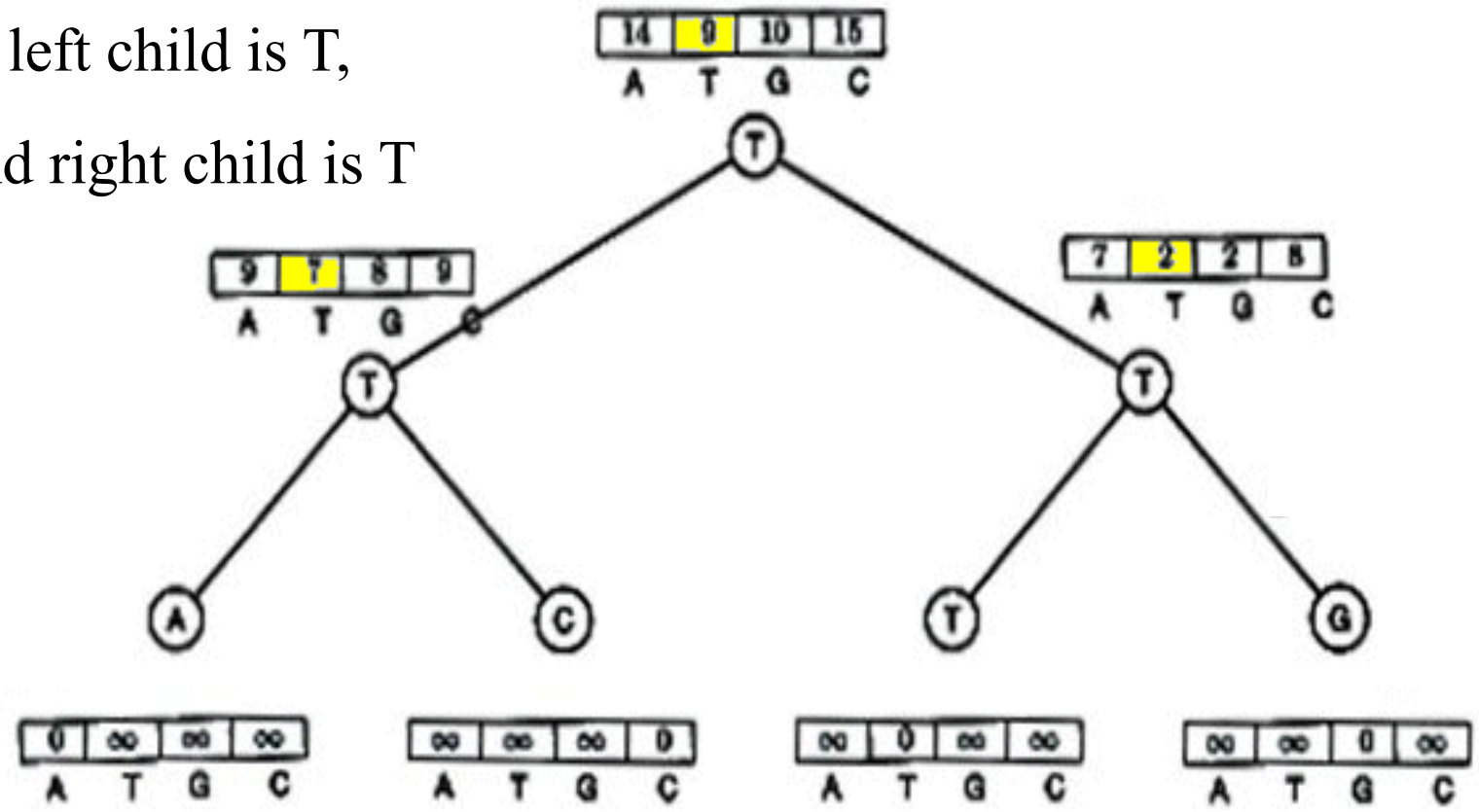
- The scores at the root vertex have been computed by going up the tree
- After the scores at root vertex are computed the Sankoff algorithm moves down the tree and assign each vertex with optimal character.

Sankoff Algorithm (cont.)

9 is derived from 7 + 2

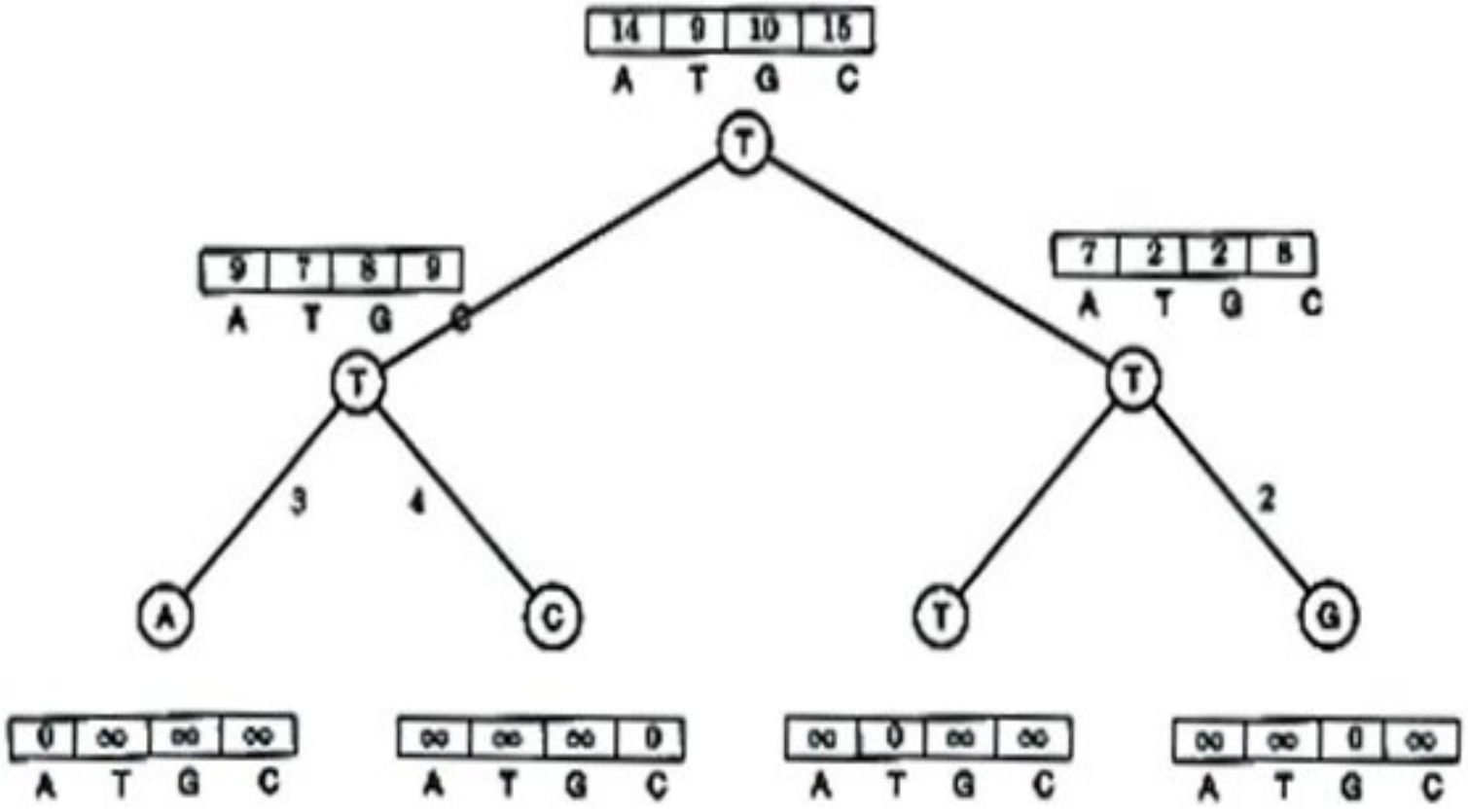
So left child is T,

And right child is T



Sankoff Algorithm (cont.)

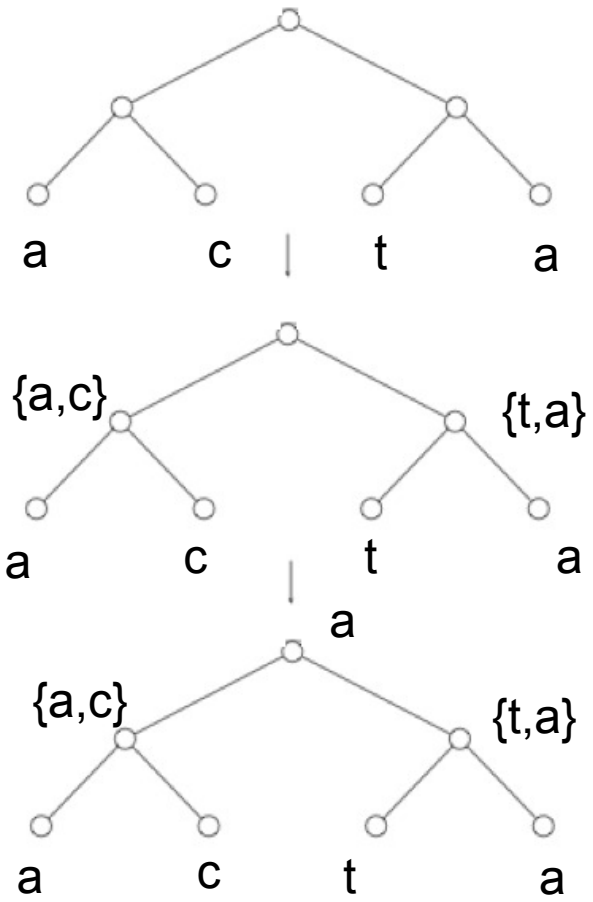
And the tree is thus labeled…



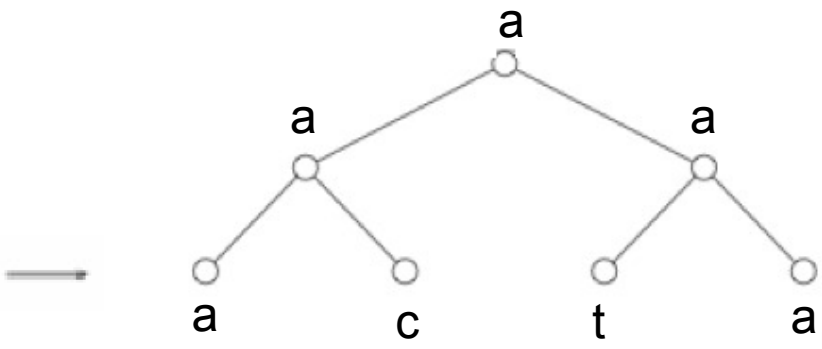
Fitch's Algorithm

- Also solves Small Parsimony problem
- Assigns a set of characters to every vertex in the tree.
- If the two children's sets of character overlap, it's the common set (intersection) of them
- If not, it's the combined set (union) of them.

Fitch's Algorithm (cont'd)



An example:



Fitch Algorithm

1) Assign a *set of possible letters* to every vertex, traversing the tree from leaves to root

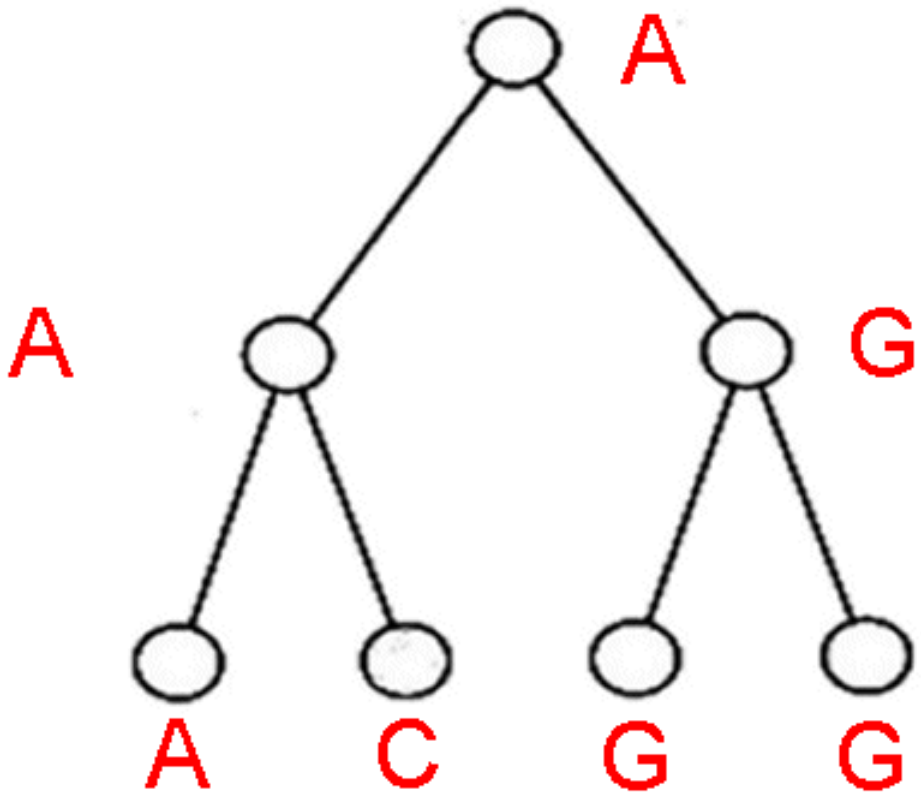
- Each node's set is the union of its children's sets (leaves contain their label) if they are disjoint
  - E.g. if the node we are looking at has a left child labeled {A} and a right child labeled {C}, the node will be given the set {A, C}
- Each node's set is the intersection of its children's sets (leaves contain their label) if they overlap
  - E.g. if the node we are looking at has a left child labeled {A, C} and a right child labeled {A, T}, the node will be given the set {A}

Fitch Algorithm (cont.)

2) Assign **labels** to each vertex, traversing the tree from root to leaves

- Assign root arbitrarily from its set of letters
- For all other vertices, if its parent's label is in its set of letters, assign it its parent's label
- Else, choose an arbitrary letter from its set as its label

Fitch Algorithm (cont.)

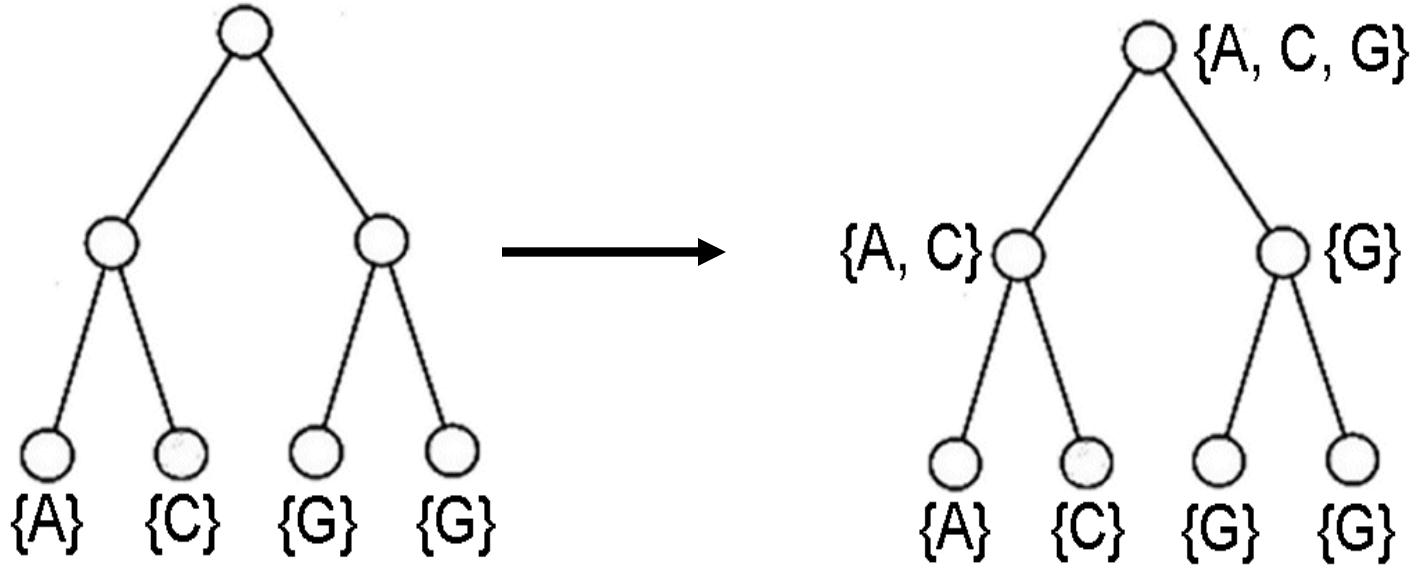


Fitch vs. Sankoff

- Both have an O(*nk*) runtime
- Are they actually different?
- Let's compare …

Fitch

As seen previously:



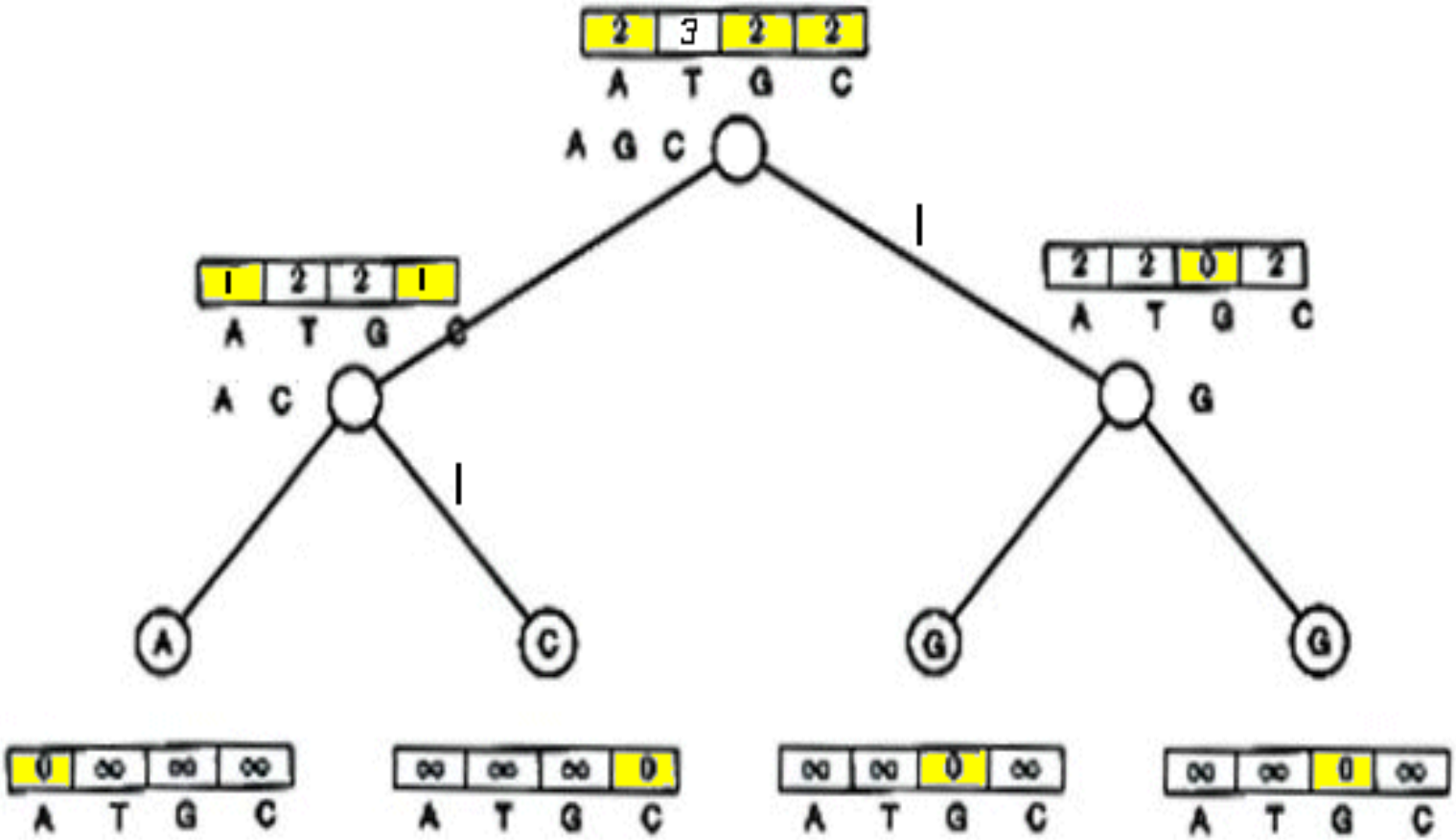
Comparison of Fitch and Sankoff

- As seen earlier, the scoring matrix for the Fitch algorithm is merely:

	A	T	G	C
A	0	1	1	1
T	1	0	1	1
G	1	1	0	1
C	1	1	1	0

- So let's do the same problem using Sankoff algorithm and this scoring matrix

Sankoff


2 3 2 2
A T G C
A G C
1
1 2 2 1
A T G
A C
1
2 2 0 2
A T G C
G
A
C
G
G
0 ∞ ∞ ∞
A T G C
∞ ∞ ∞ 0
A T G C
∞ ∞ 0 ∞
A T G C
∞ ∞ 0 ∞
A T G C

Sankoff vs. Fitch

- The Sankoff algorithm gives the **same** set of **optimal** labels as the Fitch algorithm
- For Sankoff algorithm, character t is *optimal* for vertex v if $s_t(v) = \min_{1 \le i \le k} s_i(v)$
  - Denote the set of optimal letters at vertex v as $S(v)$
    - If $S(left\ child)$ and $S(right\ child)$ overlap, assign $S(parent)$ is the intersection
    - else assign $S(parent)$ the union of $S(left\ child)$ and $S(right\ child)$
    - This is also the Fitch recurrence
- The two algorithms are **identical**

Large Parsimony Problem

- Input: An n x 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 such that the parsimony score is minimized over all possible trees and all possible labelings of internal vertices

No tree is provided. So we have to infer both the tree and the ancestor characters

Large Parsimony Problem (cont.)

- Possible search space is huge, especially as n increases
- How many rooted binary trees with n leafs?

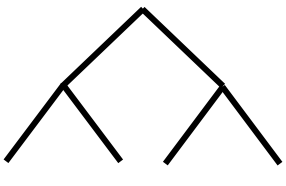
$$T(n) = \frac{(2n-3)!}{2^{n-2}(n-2)!}$$

- $T(n)$ for 2, 3, 4, 5, 6, 7, 8, 9, 10, …

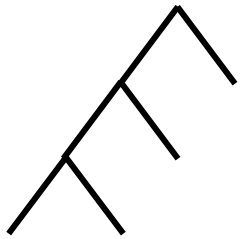
  1, 3, 15, 105, 945, 10395, 135135, 2027025, 34459425…

Large Parsimony Problem (cont.)

- e.x. 4 leaf trees



$$\frac{1}{2}\binom{4}{2} = 3$$



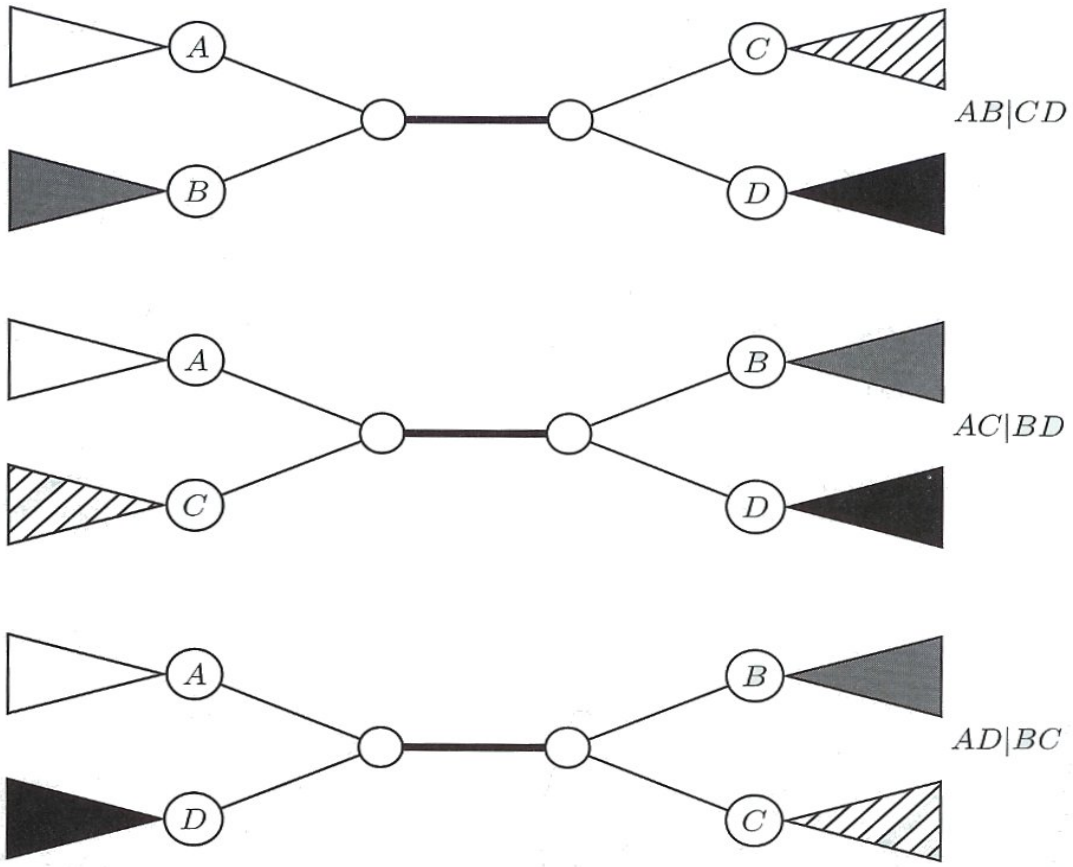
$$4 \times 3 = 12$$

- Problem is NP-complete
  - Exhaustive search only possible w/ small $n(< 10)$
- Hence, branch and bound or heuristics used

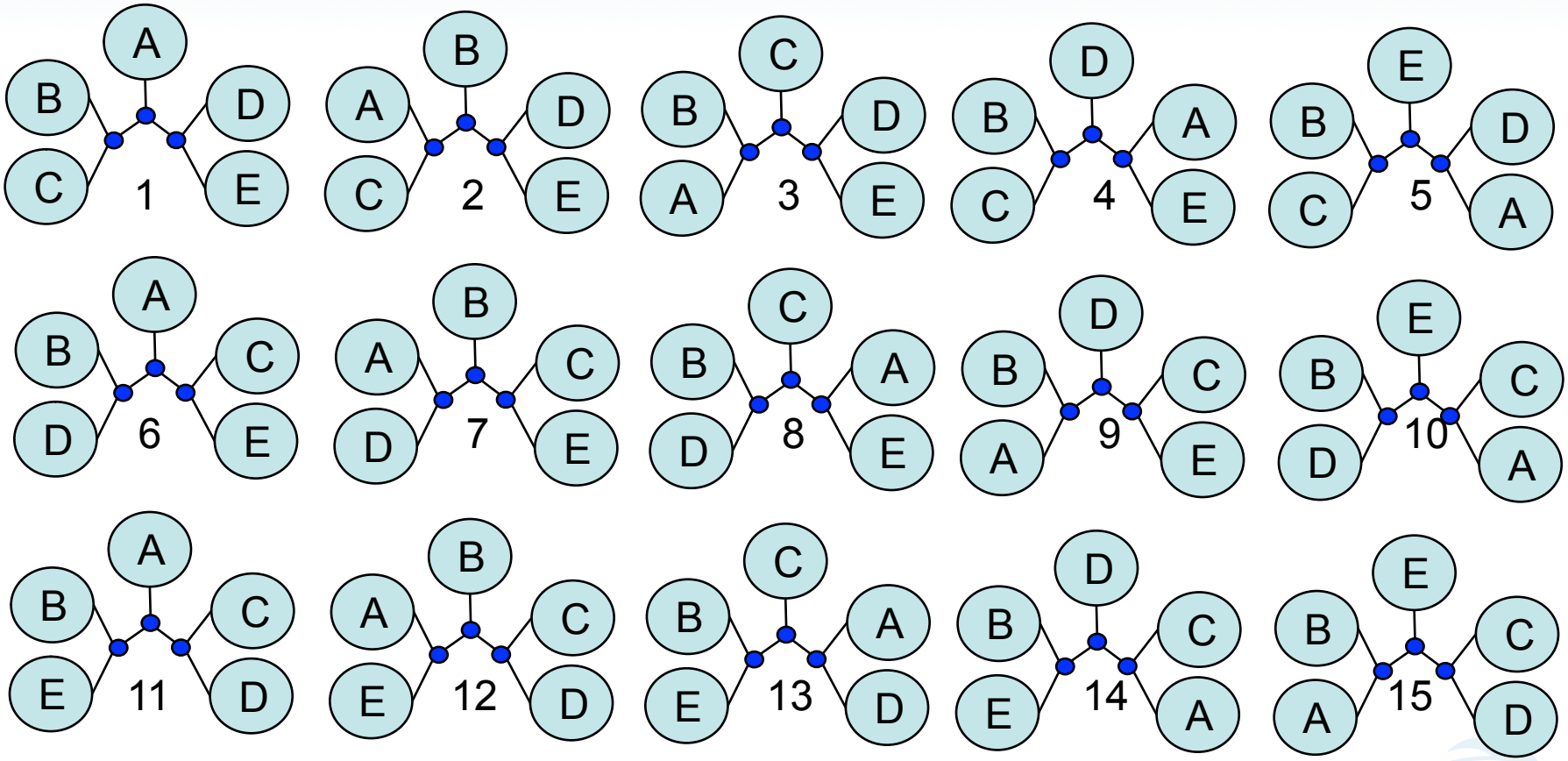
Nearest Neighbor Interchange
A Greedy Algorithm

- A Branch Swapping algorithm
- Only evaluates a subset of all possible trees
- Defines a *neighbor* of a tree as one reachable by a *nearest neighbor interchange*
  - A rearrangement of the four subtrees defined by one internal edge
  - Only three different rearrangements per edge

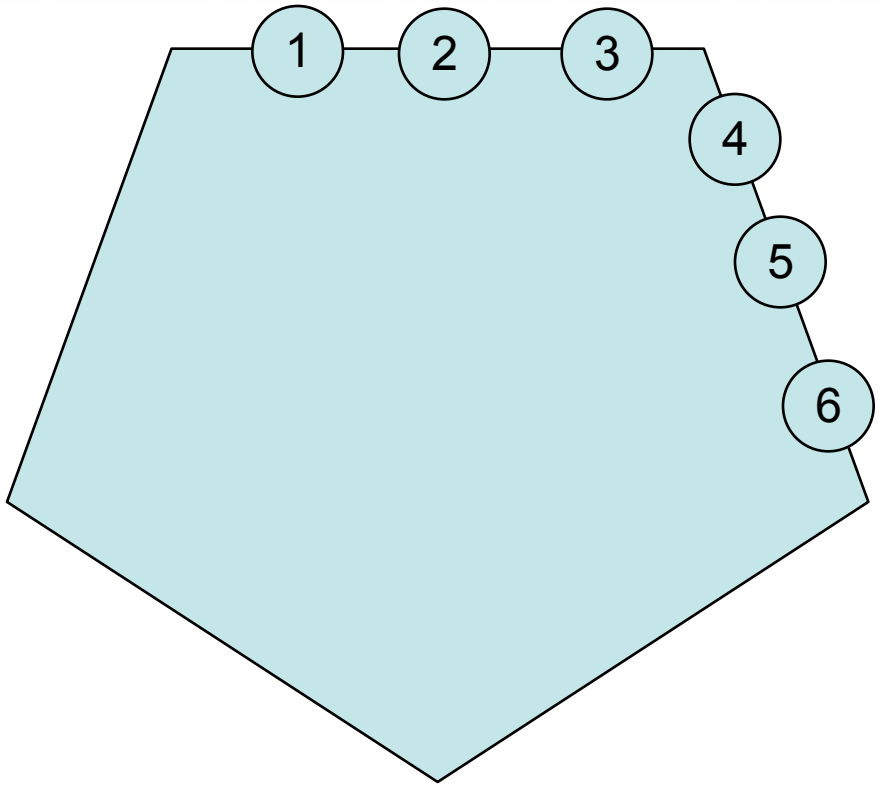
Nearest Neighbor Interchange (cont.)



Neighbors of all 5-leaf Trees



Neighbors of 5-leaf Trees



Nearest Neighbor Interchange (cont.)

- Start with an arbitrary tree and check its neighbors
- Move to a neighbor if it provides the best improvement in parsimony score
- No way of knowing if the result is the **most** parsimonious tree
- Could be stuck in local optimum

Next Time

- Are Perfect Phylogeny Trees possible?
- Under what conditions can we construct a tree that explains every mutation in the most parsimonious way?