CSCI 5481: Computational Techniques in Genomics (Fall 2008)

# Phylogeny

**Rui Kuang**

**Department of Computer Science and Engineering**
**University of Minnesota**
**kuang@cs.umn.edu**

UNIVERSITY OF MINNESOTA
Twin Cities • Duluth • Morris • Crookston • Rochester • Other Locations

## Parsimony – What if we don’t have distances

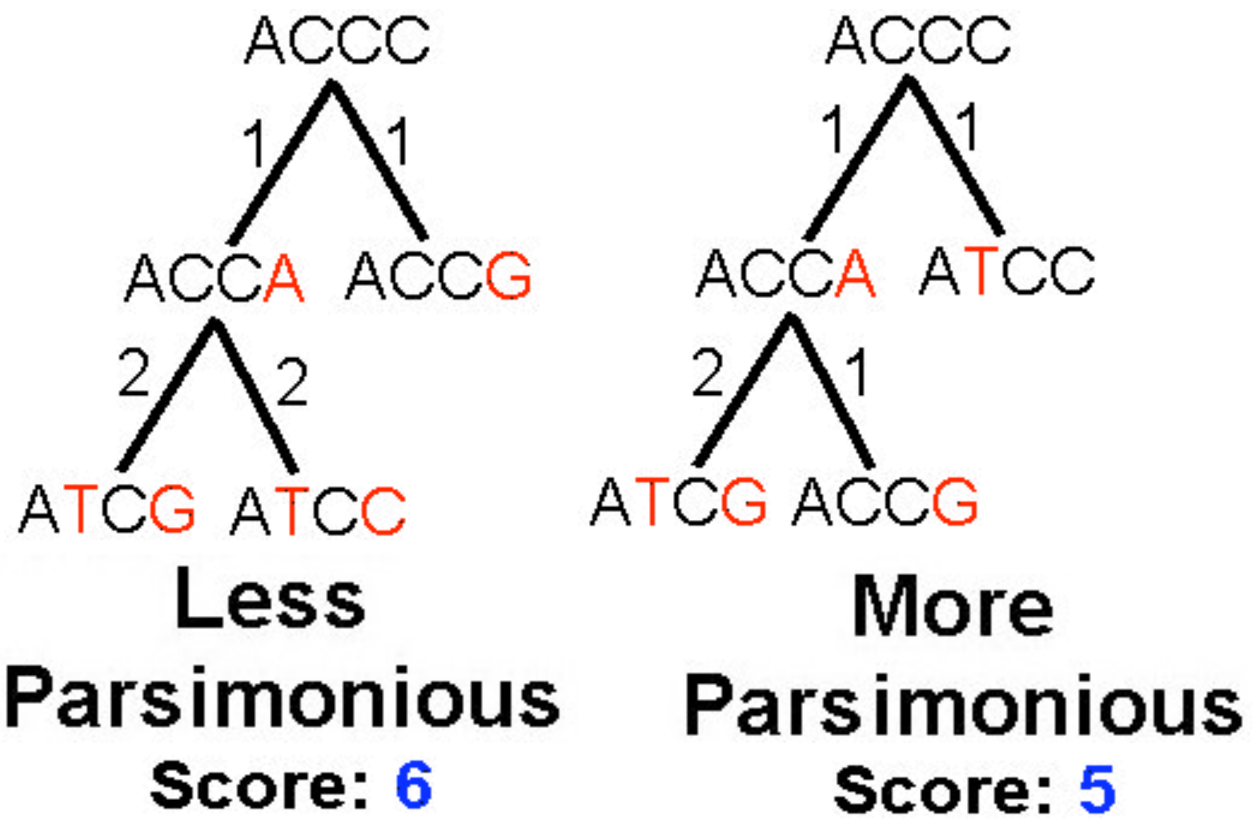
**Idea:**

Find the tree that explains the observed sequences with a minimal number of substitutions

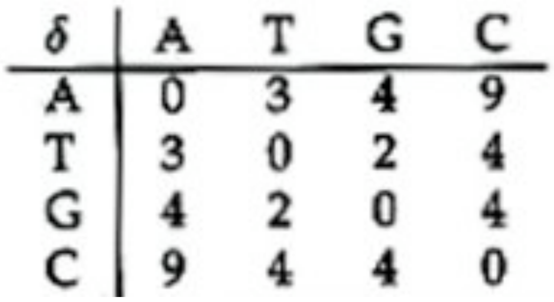
**Two computational subproblems:**

1. Find the parsimony cost of a given tree (easy): small problem
2. Search through all tree topologies (**hard**): large problem

## Parsimony and Tree Reconstruction



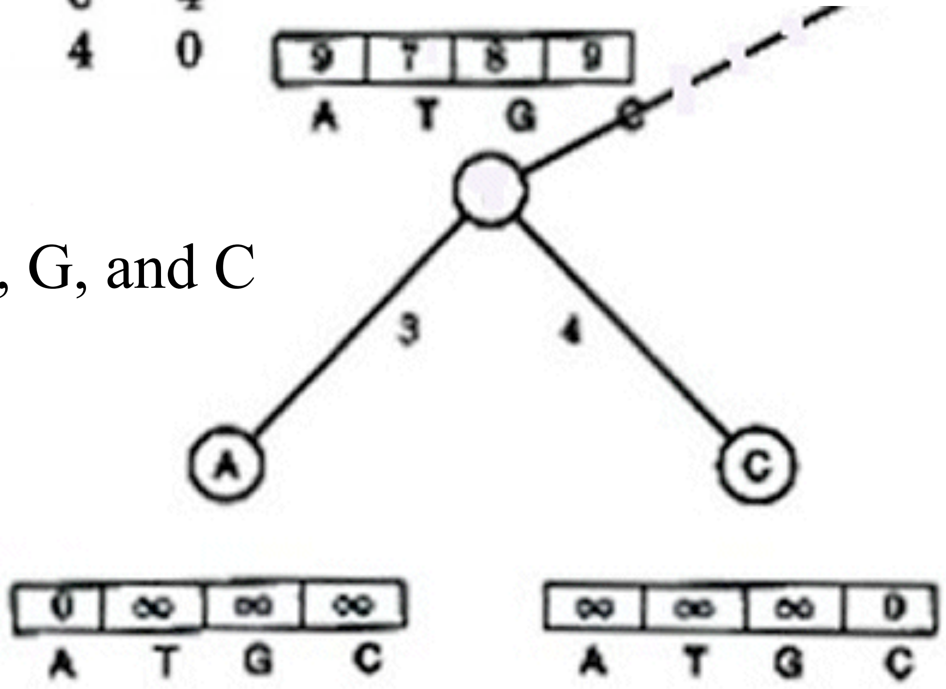
# Sankoff Algorithm (cont.)



| 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 |

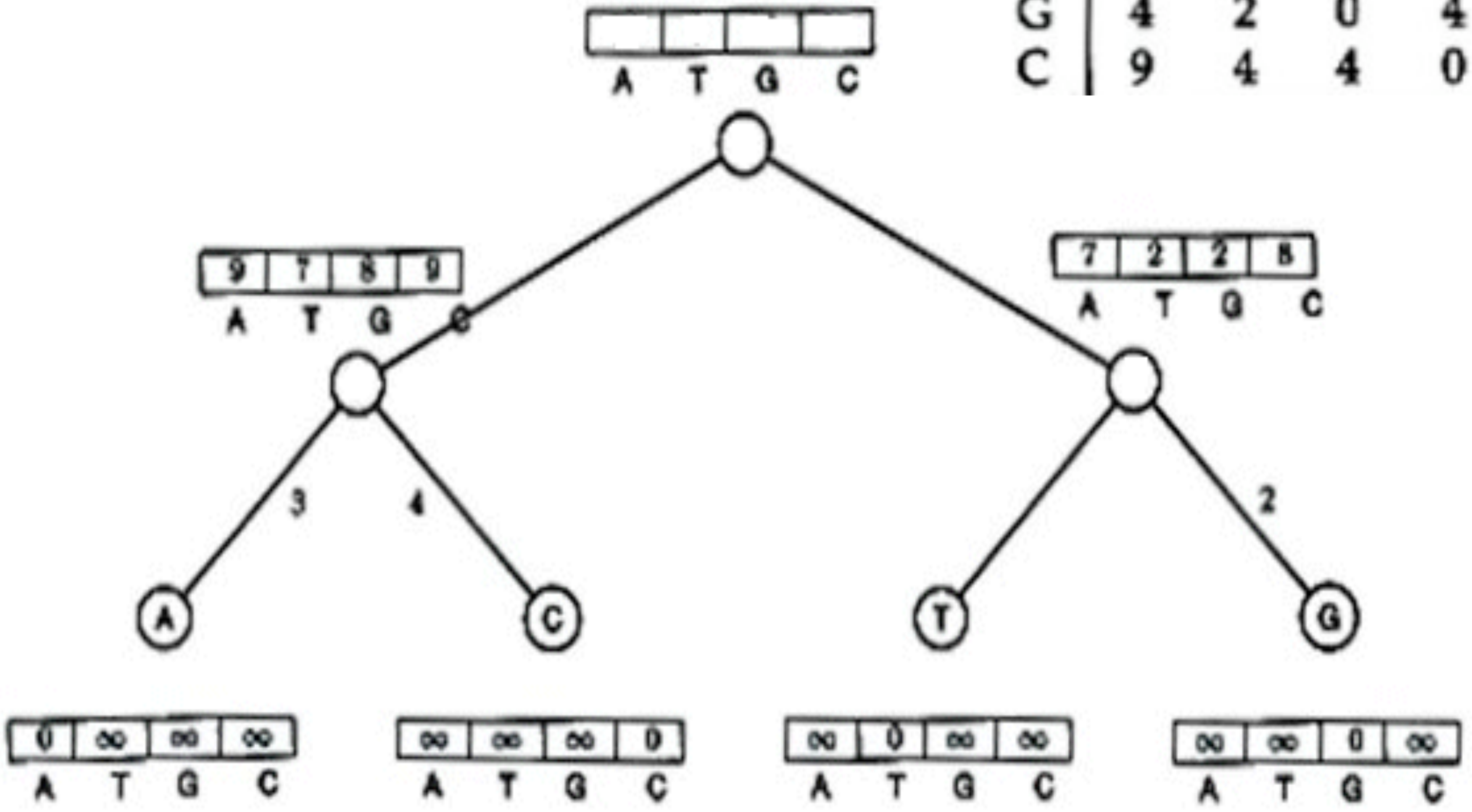
$$s_t(v) = \min_i \{s_i(u) + \delta_{i,\,t}\} + \min_j\{s_j(w) + \delta_{j,\,t}\}$$

Repeat for T, G, and C

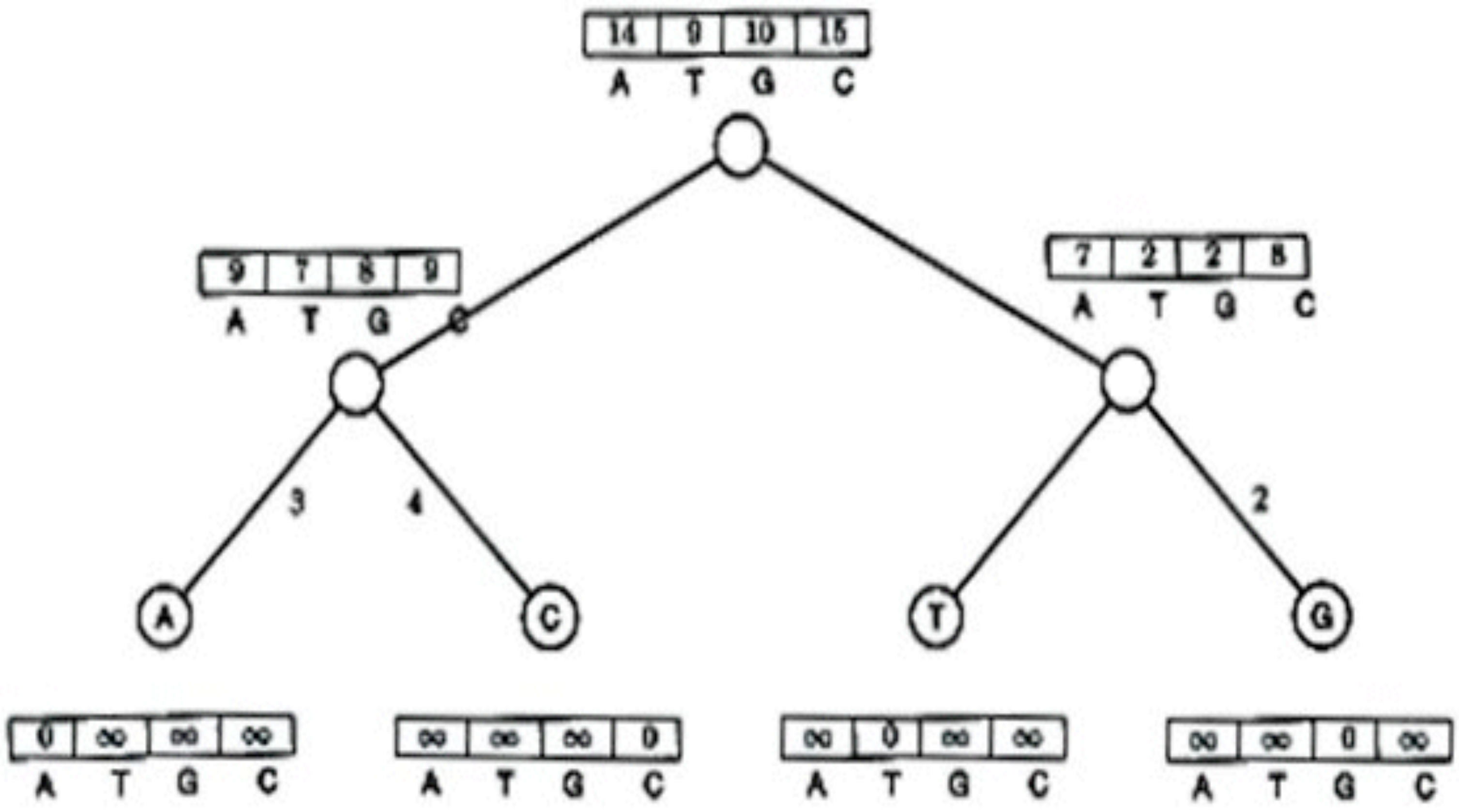


## Sankoff Algorithm (cont.)

| 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 |



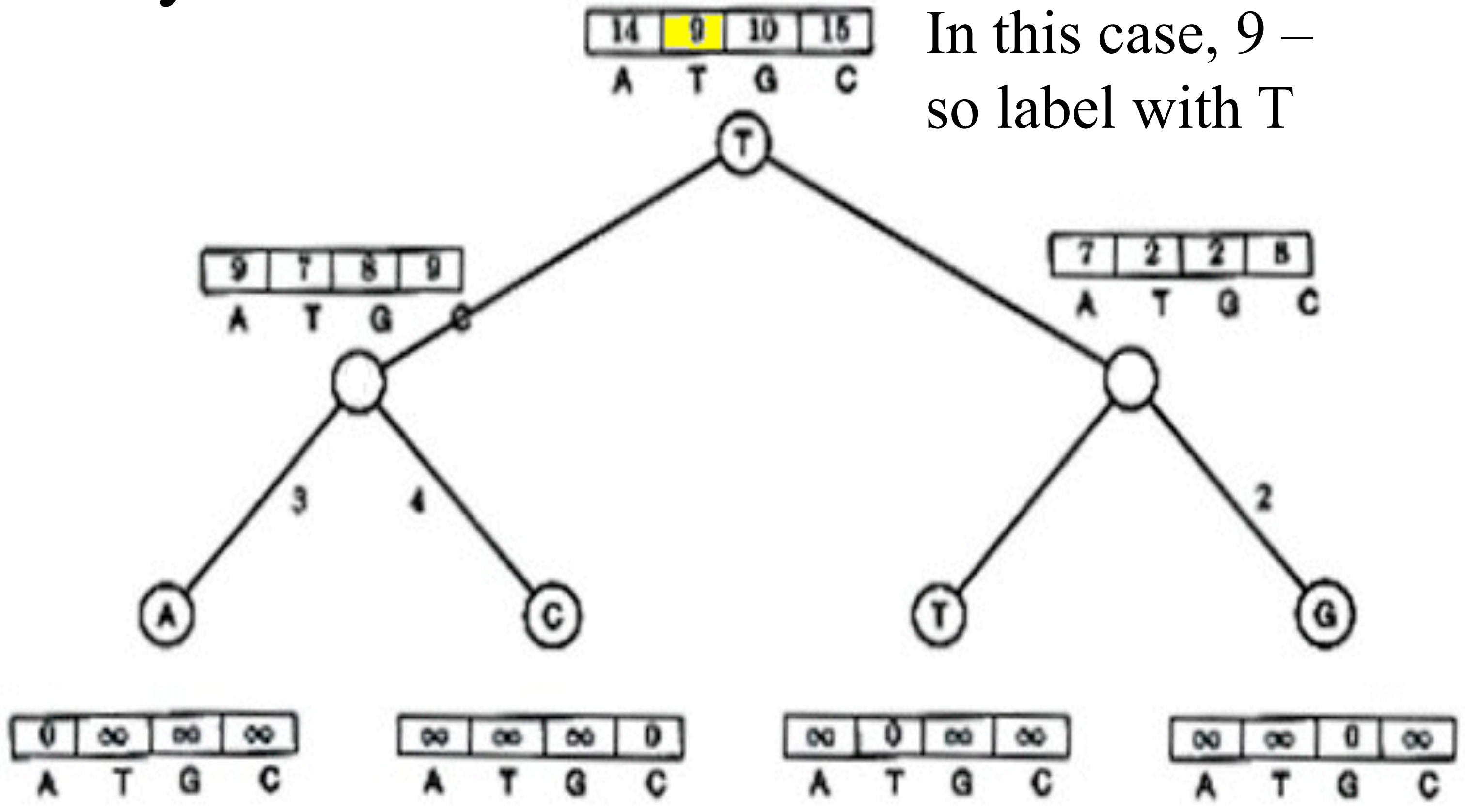
# Sankoff Algorithm (cont.)



## 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

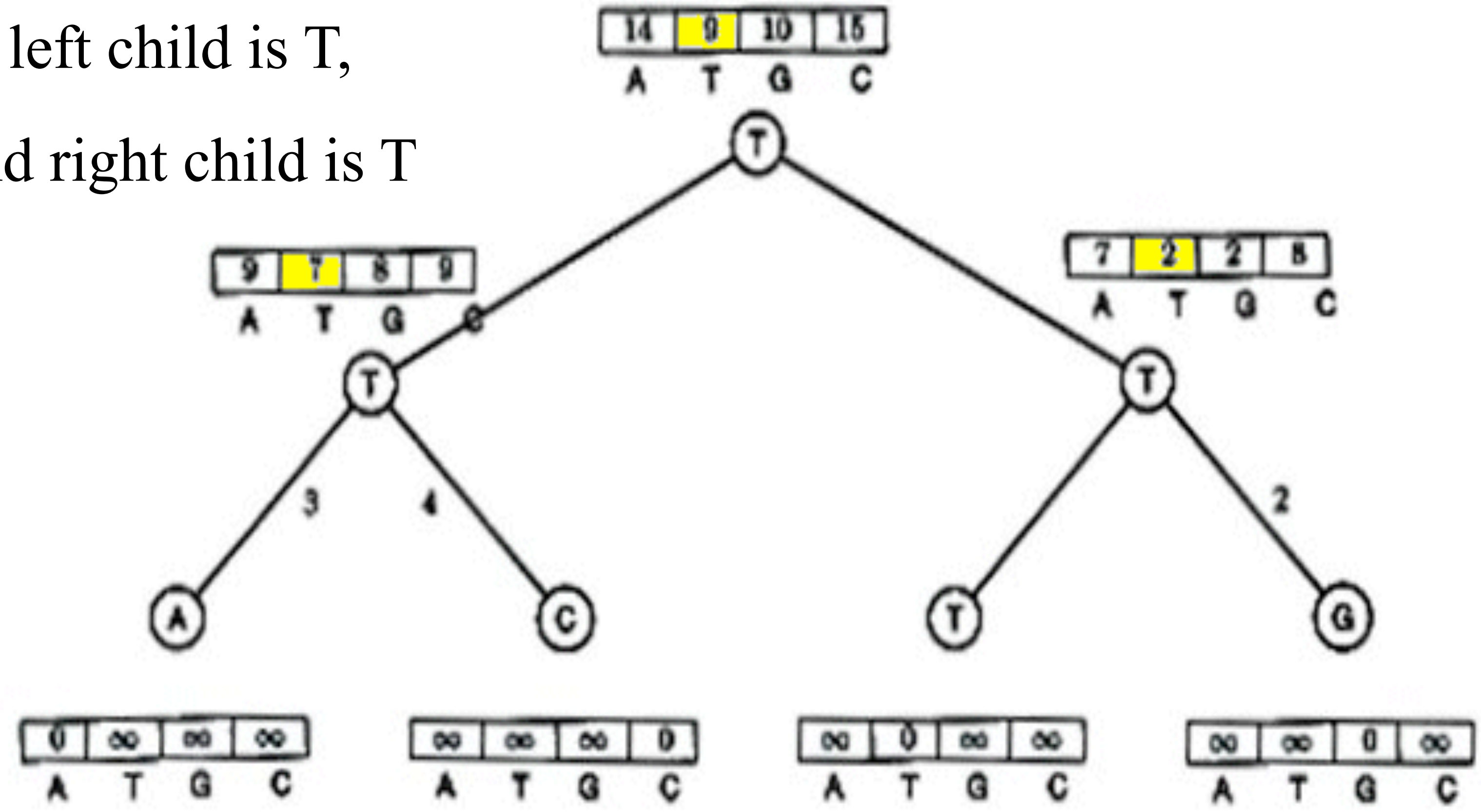
- The scores at the root vertex have been computed by going up the tree
- After the scores at root vertex are computed the Sankoff algorithm moves down the tree and assign each vertex with optimal character.

# Sankoff Algorithm (cont.)

9 is derived from 7 + 2

So left child is T,

And right child is T

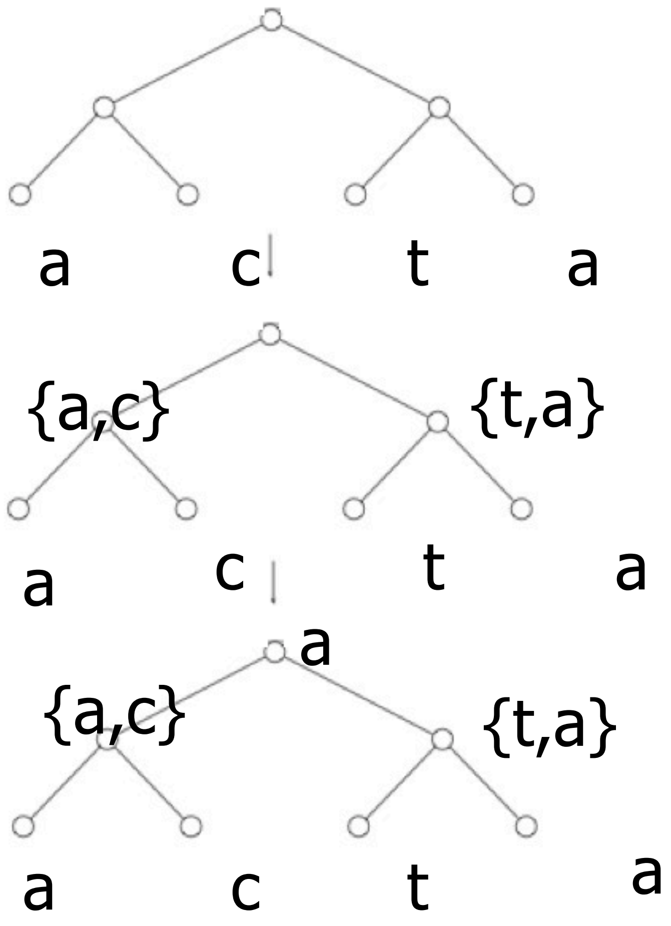


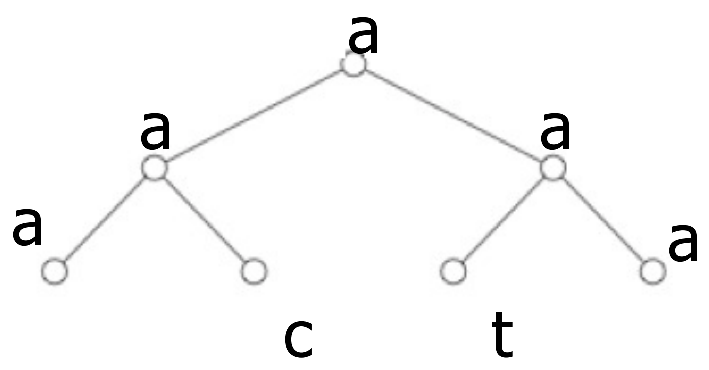
## Fitch’s Algorithm

- Solves Small Parsimony problem
- Dynamic programming in essence
- Assigns a set of letter to every vertex in the tree.
- If the two children’s sets of character overlap, it’s the common set of them
- If not, it’s the combined set 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

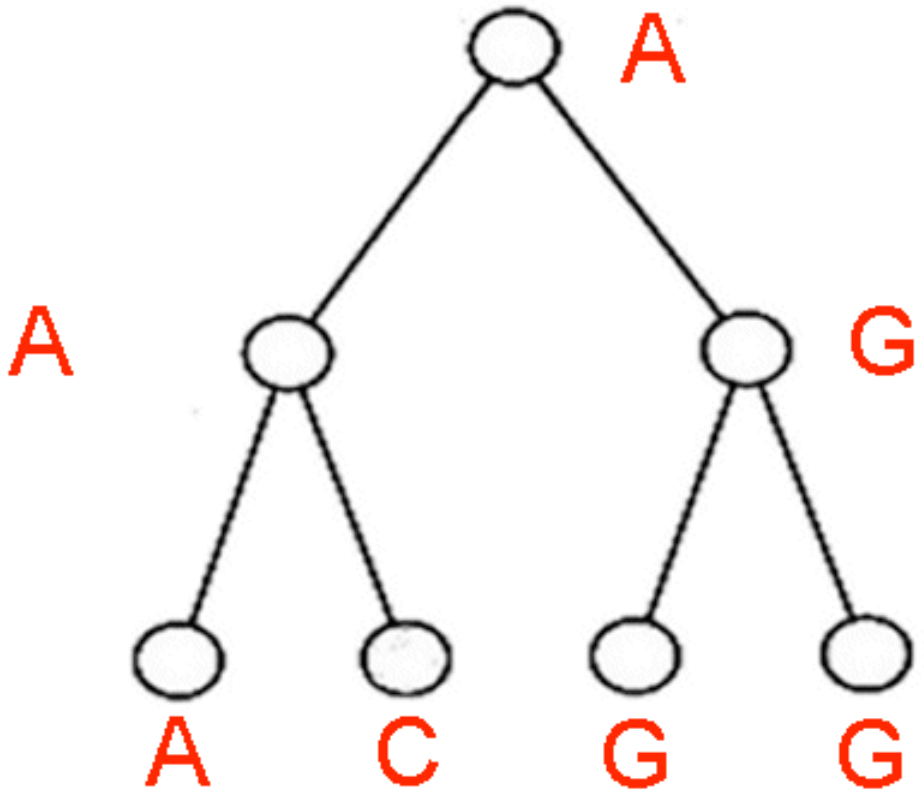
- Each node's set is the union/intersection of its children's sets
  - E.g. if the node we are looking at has a left child labeled {A, C} and a right child labeled {A, T}, the node will be given the set {A, C, T}

## 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.)

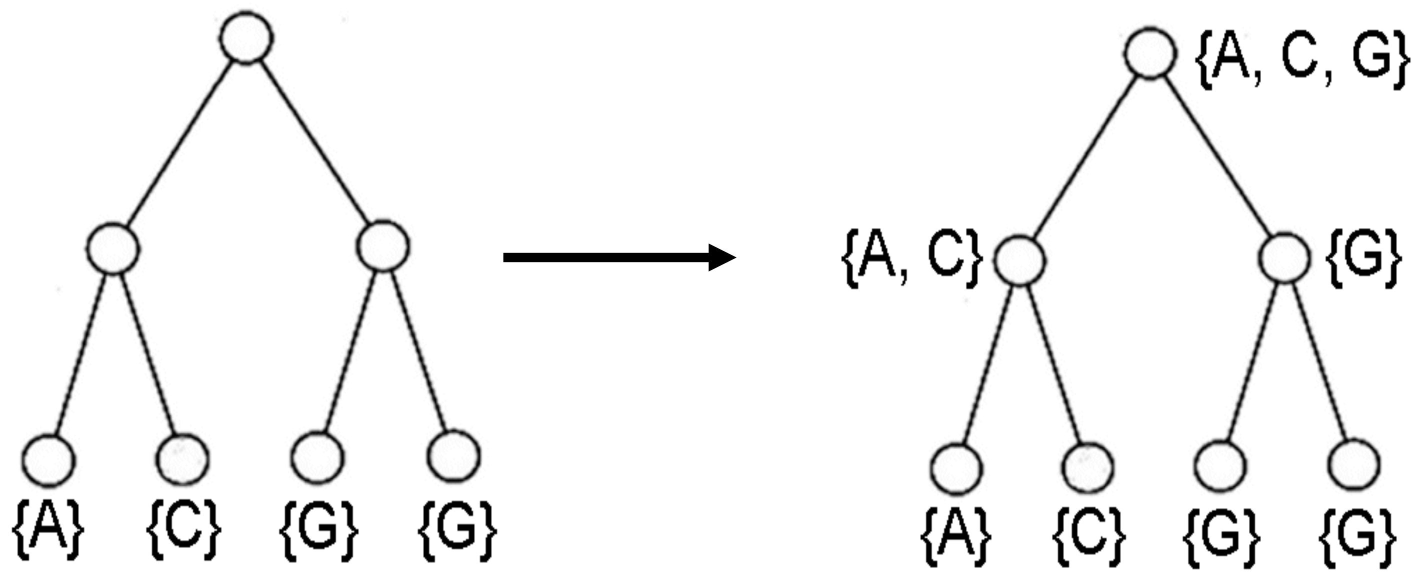


## Fitch vs. Sankoff

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

# Fitch

As seen previously:



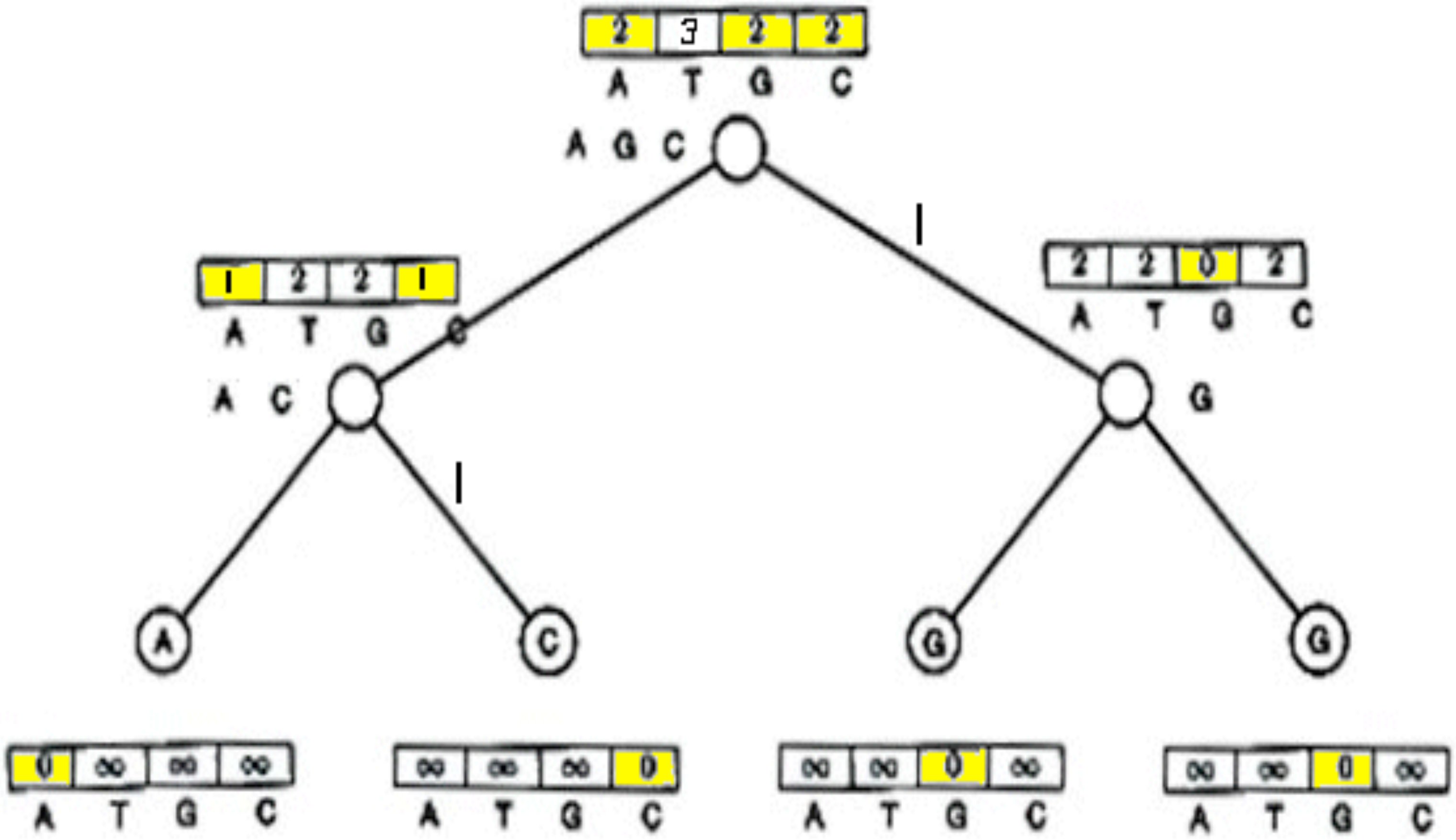
## Comparison of Fitch and Sankoff

- 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 C
A C
2 2 0 2
A T G C
G
1
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, *S*(*parent*) is the intersection
    - Else it's 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

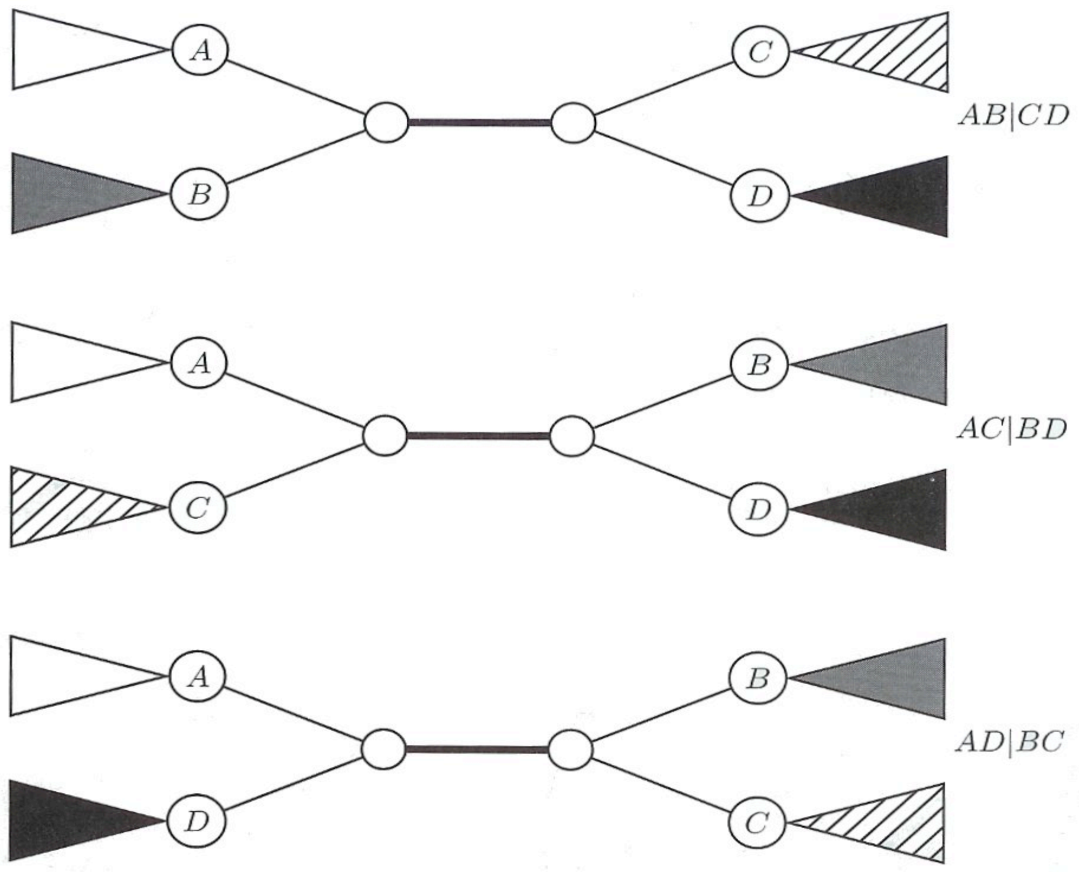
## Large Parsimony Problem (cont.)

- Possible search space is huge, especially as *n* increases
  - Almost (2*n* – 5)!! possible unrooted trees
  - 10 sequences, 2,027,025
- 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.)

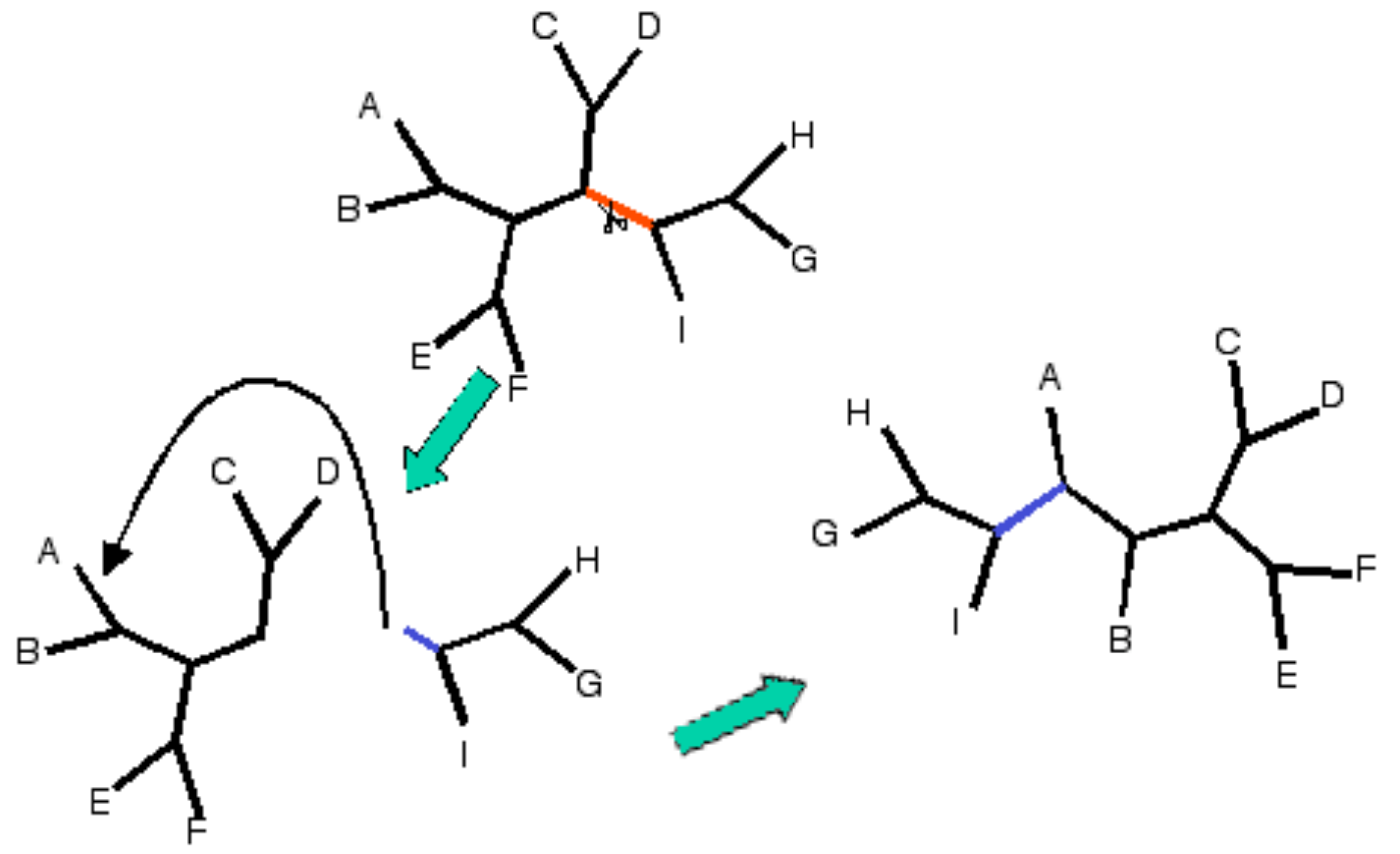


## Nearest Neighbor Interchange (cont.)

- 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

# Subtree Pruning and Regrafting

## Another Branch Swapping Algorithm

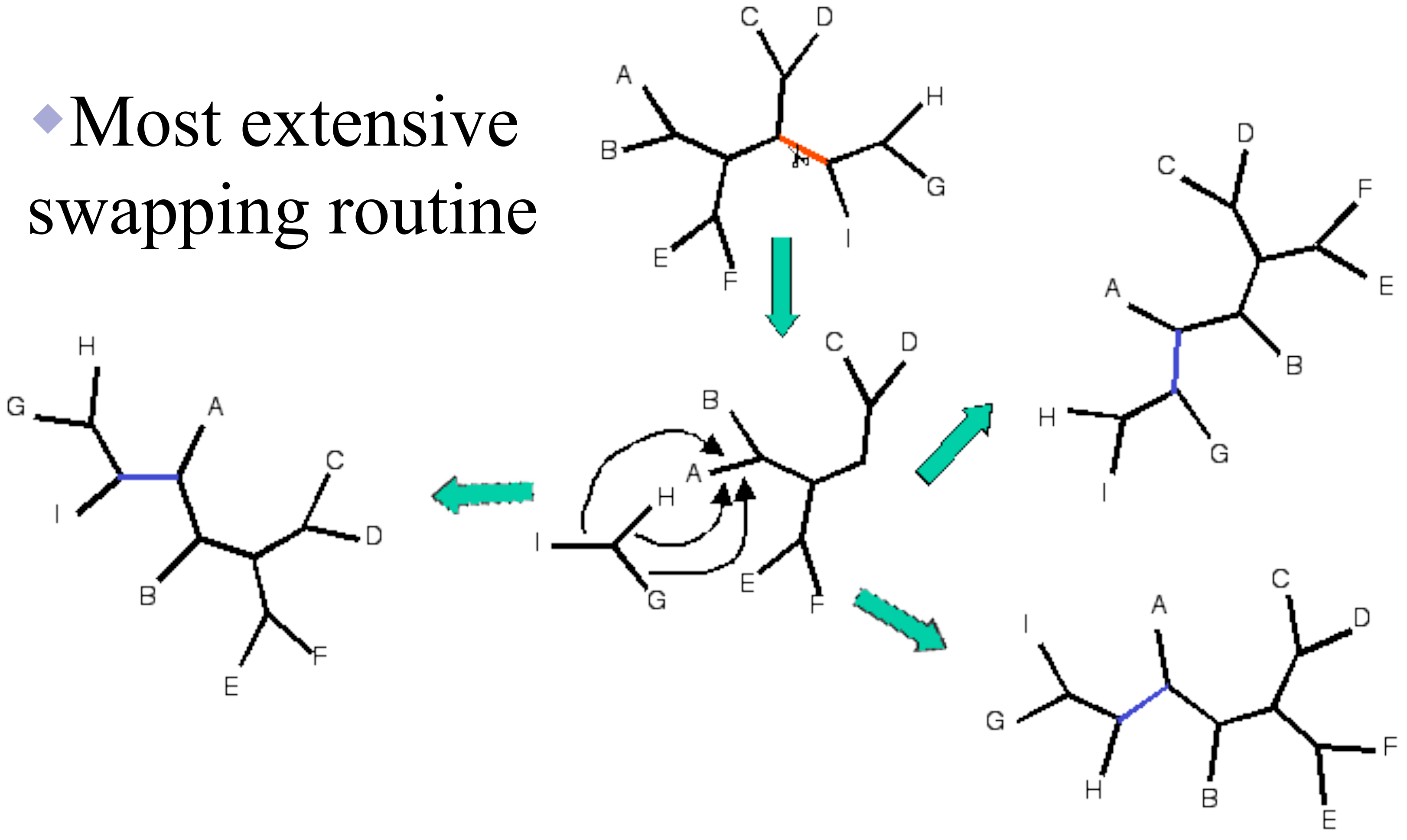


http://artedi.ebc.uu.se/course/BioInfo-10p-2001/Phylogeny/Phylogeny-TreeSearch/SPR.gif

# Tree Bisection and Reconnection

Another Branch Swapping Algorithm

- Most extensive swapping routine



## Parsimony – What if we don’t have distances

Idea:

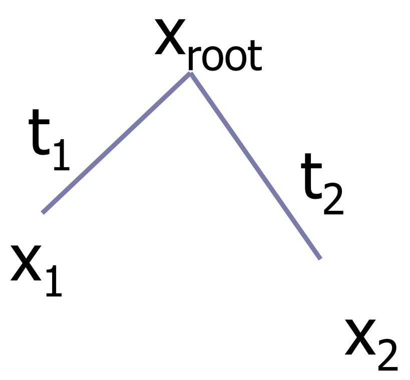
Find the tree that explains the observed sequences with a minimal number of substitutions

**Two computational subproblems:**

1. Find the parsimony cost of a given tree (easy): small problem

2. Search through all tree topologies (**hard**): large problem

# Probabilistic Methods



$$P(j \mid i,t) = \begin{cases} \frac{1}{4}(1 + 3e^{-4\alpha t}) \text{ for j} = i \\ \frac{1}{4}(1 - e^{-4\alpha t}) \text{ for } j \neq i \end{cases}$$

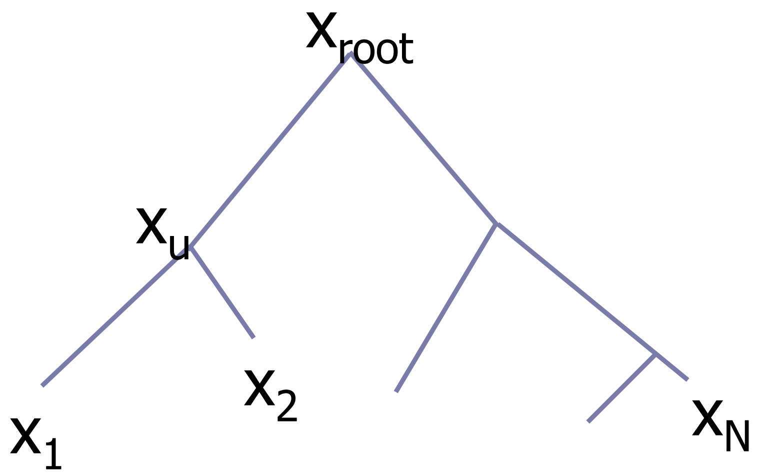
A more refined measure of evolution along a tree than parsimony

$$P(x_1, x_2, x_{root} \mid t_1, t_2) = P(x_{root})\, P(x_1 \mid t_1, x_{root})\, P(x_2 \mid t_2, x_{root})$$

If we use Jukes-Cantor, for example, and $x_1 = x_{root} = A$, $x_2 = C$, $t_1 = t_2 = 1$,

$$= p_A \times ¼(1 + 3e^{-4\alpha}) \times ¼(1 - e^{-4\alpha}) = (¼)^3(1 + 3e^{-4\alpha})(1 - e^{-4\alpha})$$

## Probabilistic Methods



- If we know all internal labels $x_u$,

$$P(x_1, x_2, \ldots, x_N, x_{N+1}, \ldots, x_{2N-1} \mid T, t) = P(x_{root}) \prod_{j \neq root} P(x_j \mid x_{parent(j)}, t_{j,\, parent(j)})$$

- Usually we don't know the internal labels, therefore

$$P(x_1, x_2, \ldots, x_N \mid T, t) = \sum_{x_{N+1}} \sum_{x_{N+2}} \ldots \sum_{x_{2N-1}} P(x_1, x_2, \ldots, x_{2N-1} \mid T, t)$$

# Felsenstein's Likelihood Algorithm

To calculate $P(x_1, x_2, \ldots, x_N \mid T, t)$

Let $P(L_k \mid a)$ denote the prob. of all the leaves below node k, given that the residue at k is a

**Initialization:**

Set $k = 2N - 1$

**Iteration:** Compute $P(L_k \mid a)$ for all $a \in \Sigma$

```
If k is a leaf node:
        Set $P(L_k \mid a) = \mathbf{1}(a = x_k)$
If k is not a leaf node:
        1. Compute $P(L_i \mid b)$, $P(L_j \mid b)$ for all b, for daughter nodes i, j
        2. Set $P(L_k \mid a) = \sum_{b,c} P(b \mid a, t_i)\, P(L_i \mid b)\, P(c \mid a, t_j)\, P(L_j \mid c)$
```

**Termination:**

Likelihood at this column = $P(x_1, x_2, \ldots, x_N \mid T, t) = \sum_a P(L_{2N-1} \mid a) P(a)$

# Probabilistic Methods

Given M (ungapped) alignment columns of N sequences,

- Define likelihood of a tree:

$$L(T, t) = P(\text{Data} \mid T, t) = \prod_{m=1\ldots M} P(x_{1m}, \ldots, x_{nm}, T, t)$$

Maximum Likelihood Reconstruction:

- Given data $X = (x_{ij})$, find a topology **T** and length vector **t** that maximize likelihood L(**T**, **t**)

# Current popular methods

*HUNDREDS of programs available!*

http://evolution.genetics.washington.edu/phylip/software.html#methods

Some recommended programs:

- Discrete—Parsimony-based
  - Rec-1-DCM3
    http://www.cs.utexas.edu/users/tandy/mp.html
    Tandy Warnow and colleagues
- Probabilistic
  - SEMPHY
    http://www.cs.huji.ac.il/labs/compbio/semphy/
    Nir Friedman and colleagues