Parsimony-Based Approaches to Inferring Phylogenetic Trees

BMI/CS 576

www.biostat.wisc.edu/bmi576.html

Colin Dewey

cdewey@biostat.wisc.edu

Fall 2009

Phylogenetic Tree Approaches

- three general types
  - *distance*: find tree that accounts for estimated evolutionary distances
  - *parsimony*: find the tree that requires minimum number of changes to explain the data
  - *maximum likelihood*: find the tree that maximizes the likelihood of the data

Parsimony Based Approaches

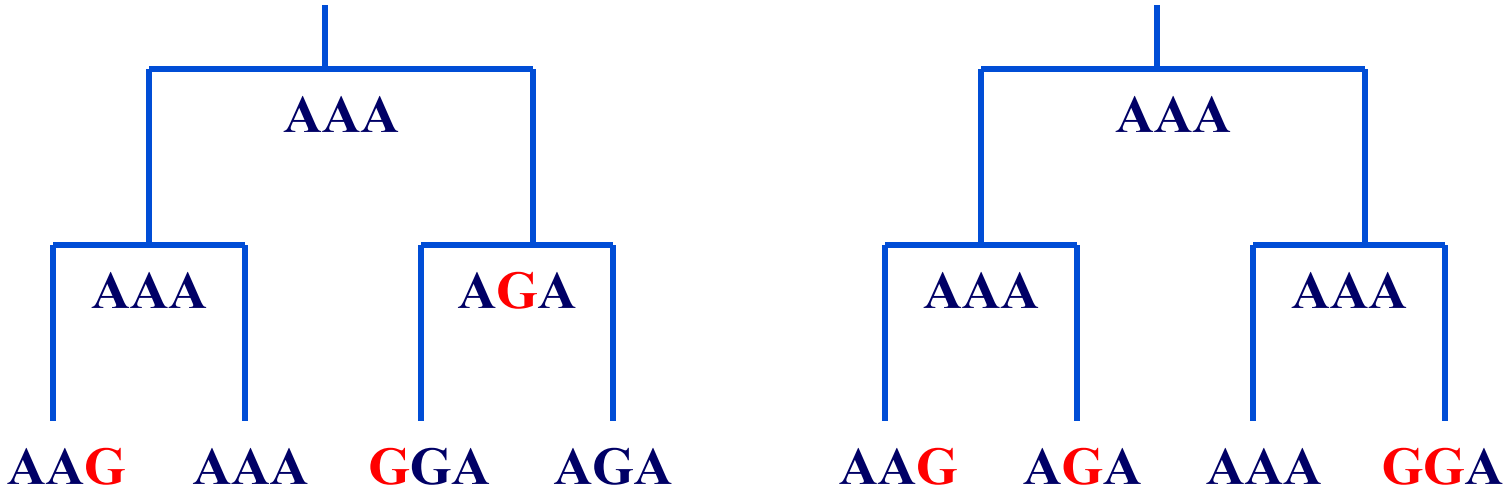
given: character-based data

do: find tree that explains the data with a minimal number of changes

- focus is on finding the right tree topology, not on estimating branch lengths

Parsimony Example

- there are various trees that could explain the phylogeny of the sequences AAG, AAA, GGA, AGA including these two:



- parsimony prefers the first tree because it requires fewer substitution events

Parsimony Based Approaches

- usually these approaches involve two separate components
  1. a procedure to find the minimum number of changes needed to explain the data (for a given tree topology)
  2. a search through the space of trees

Finding Minimum Number of Changes for a Given Tree

- Basic assumptions:
  - any state (e.g. nucleotide, amino acid) can convert to any other state
  - the “costs” of these changes are uniform
  - assumes positions are independent
    - Thus, we can compute the minimum number of changes for each position separately

Finding Minimum Number of Changes for a Given Tree

- Brute force method:
  - For each possible assignment of states to the internal nodes
    - Calculate number of changes for that assignment
  - Report the minimum number of changes found

- Runtime: $O(k^N)$
  - k = number of possible character states (e.g., 4 for DNA)
  - N = number of leaves

Fitch's Algorithm

Fitch's algorithm [1971] :

1. traverse tree from leaves to root determining set of possible *states* (e.g. nucleotides) for each internal node

2. traverse tree from root to leaves picking ancestral states for internal nodes

Fitch's Algorithm: Step 1
Possible States for Internal Nodes

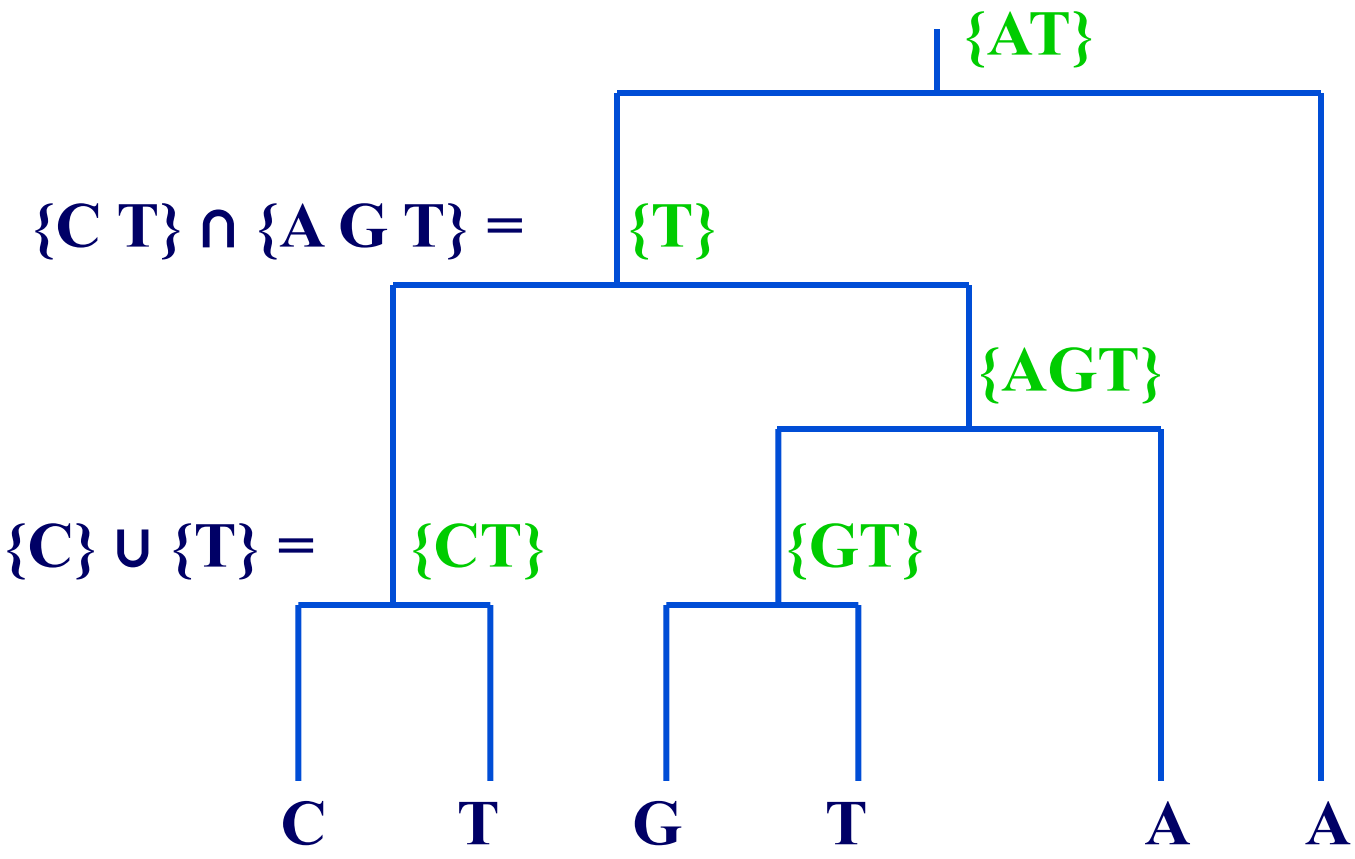
- do a *post-order* (from leaves to root) traversal of tree
- determine possible states R_i of internal node *i* with children *j* and *k*

$$R_i = \begin{cases} R_j \cup R_k, & \text{if } R_j \cap R_k = \varnothing \\ R_j \cap R_k, & \text{otherwise} \end{cases}$$

- this step calculates the number of changes required

  # of changes = # union operations

Fitch's Algorithm: Step 1 Example

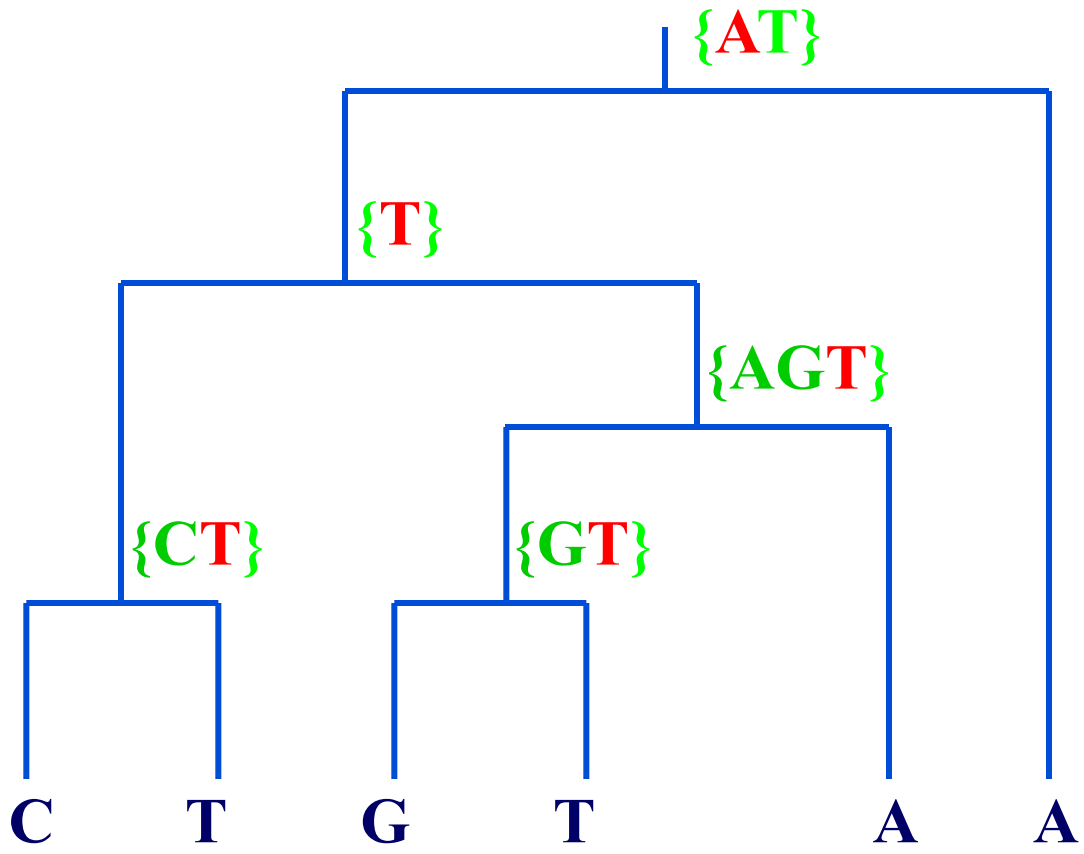


Fitch's Algorithm: Step 2
Select States for Internal Nodes

- do a *pre-order* (from root to leaves) traversal of tree
- select state r_j of internal node j with parent i

$$r_j = \begin{Bmatrix} r_i, \text{ if } r_i \in R_j \\ \text{arbitrary state} \in R_j, \text{ otherwise} \end{Bmatrix}$$

Fitch's Algorithm: Step 2



Weighted Parsimony

- [Sankoff & Cedergren, 1983]
- instead of assuming all state changes are equally likely, use different costs $S(a,b)$ for different changes
- 1st step of algorithm is to propagate costs up through tree

$a \rightarrow b$

Weighted Parsimony

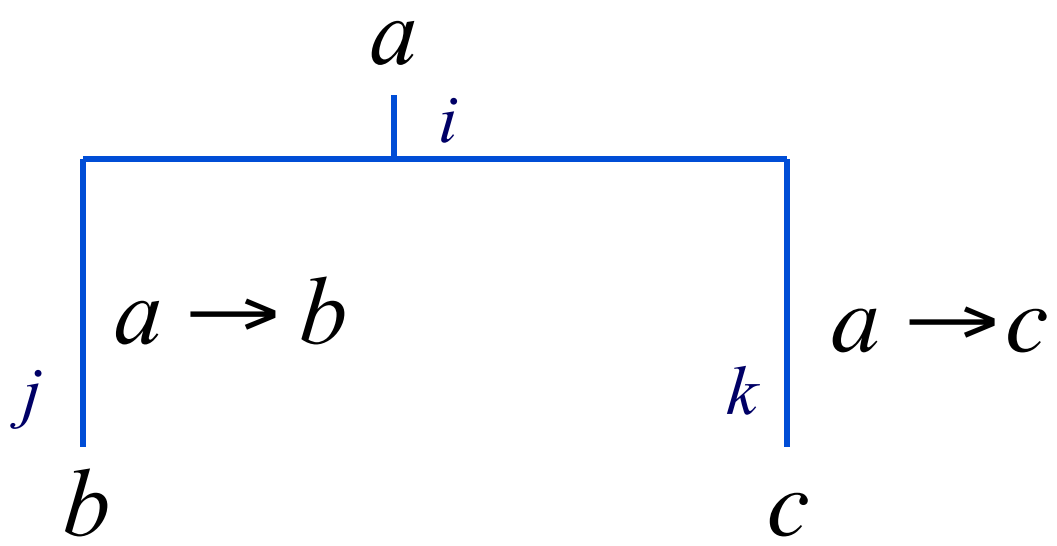
- Dynamic programming!
- Subproblem: want to determine cost $R_i(a)$ for the subtree rooted at *i* of assigning character a to node *i*
- for leaves:

$$R_i(a) = \begin{cases} 0, & \text{if } a \text{ is character at leaf} \\ \infty, & \text{otherwise} \end{cases}$$

Weighted Parsimony

- for an internal node *i* with children *j* and *k*:

$$R_i(a) = \min_b (R_j(b) + S(a,b)) + \min_c (R_k(c) + S(a,c))$$

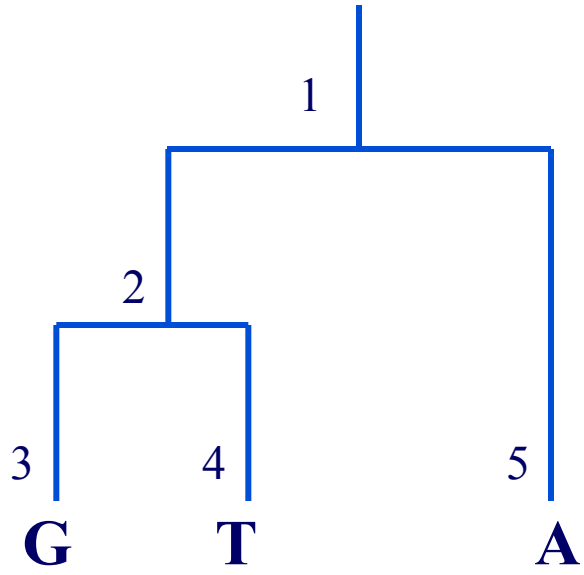


Example: Weighted Parsimony

$$R_3[A] = \infty, R_3[C] = \infty, R_3[G] = 0, R_3[T] = \infty$$

$$R_4[A] = \infty, R_4[C] = \infty, R_4[G] = \infty, R_4[T] = 0$$

$$R_2[A] = R_3[G] + S(A,G) + R_4[T] + S(A,T)$$
$$\vdots$$
$$R_2[T] = R_3[G] + S(T,G) + R_4[T] + S(T,T)$$



$$R_5[A] = 0, R_5[C] = \infty, R_5[G] = \infty, R_5[T] = \infty$$

$$R_1[A] = \min\left(R_2[A] + S(A,A), \ \ldots \ , \ R_2[T] + S(A,T)\right) + R_5[A] + S(A,A)$$
$$\vdots$$
$$R_1[T] = \min\left(R_2[A] + S(T,A), \ \ldots \ , \ R_2[T] + S(T,T)\right) + R_5[A] + S(T,A)$$

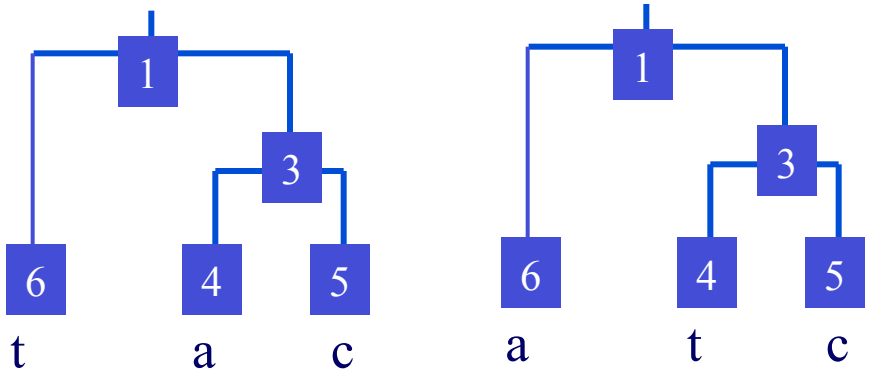
Weighted Parsimony: Step 2

- do a pre-order (from root to leaves) traversal of tree

- for root node: select minimal cost character
- for each internal node: select the character that resulted in the minimum cost explanation of the character selected at the parent

Weighted Parsimony Example

Consider the two simple phylogenetic trees shown below, and the symmetric cost matrix for assessing nucleotide changes. The tree on the right has a cost of 0.8

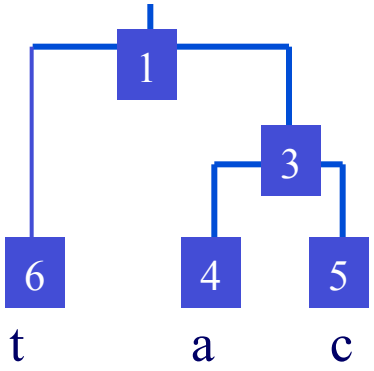


	a	c	g	t
a	0	0.8	0.2	0.9
c	0.8	0	0.7	0.5
g	0.2	0.7	0	0.1
t	0.9	0.5	0.1	0

What are the minimal cost characters for the internal nodes in the tree on the left?

Which of the two trees would the maximum parsimony approach prefer?

Weighted Parsimony Example



	a	c	g	t
a	0	0.8	0.2	0.9
c	0.8	0	0.7	0.5
g	0.2	0.7	0	0.1
t	0.9	0.5	0.1	0

$$R_3(a) = 0 + 0.8 = 0.8$$

$$R_3(c) = 0.8 + 0 = 0.8$$

$$R_3(g) = 0.2 + 0.7 = 0.9$$

$$R_3(t) = 0.9 + 0.5 = 1.4$$

$$R_1(a) = 0.9 + \min\{0.8,\ \ 0.8 + 0.8,\ \ 0.2 + 0.9,\ \ 0.9 + 1.4\} = 1.7$$

$$R_1(c) = 0.5 + \min\{0.8 + 0.8,\ \ 0.8,\ \ 0.7 + 0.9,\ \ 0.5 + 1.4\} = 1.3$$

$$R_1(g) = 0.1 + \min\{0.2 + 0.8,\ \ 0.7 + 0.8,\ \ 0.9,\ \ 0.1 + 1.4\} = 1.0$$

$$R_1(t) = 0 + \min\{0.9 + 0.8,\ \ 0.5 + 0.8,\ \ 0.1 + 0.9,\ \ 1.4\} = 1.0$$

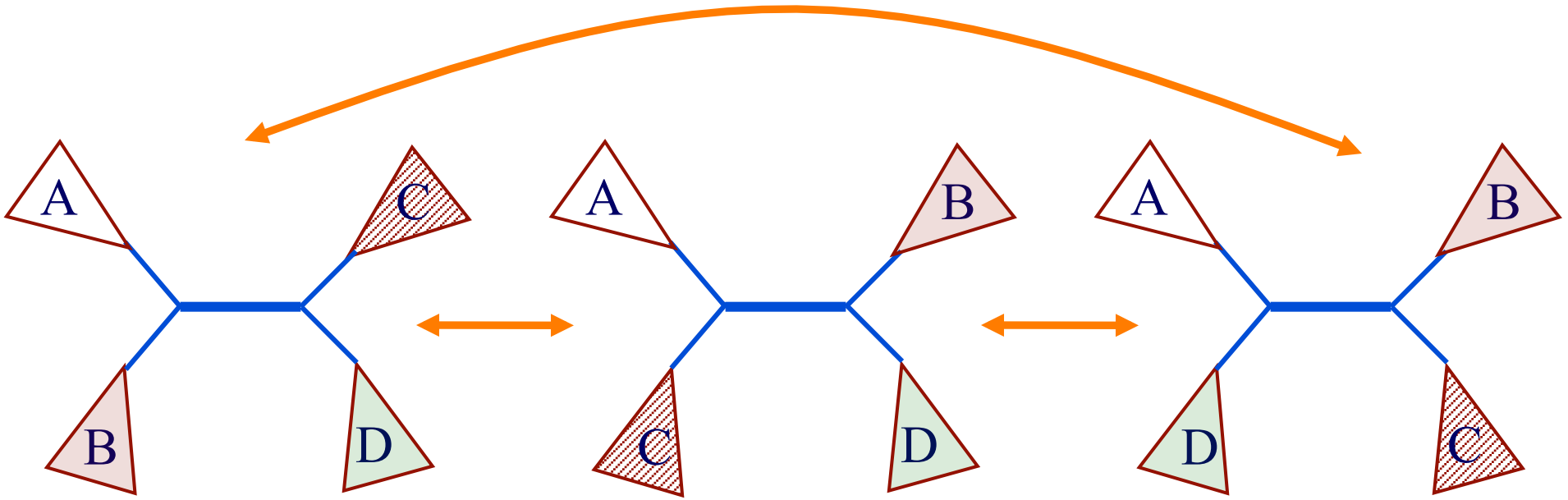
The minimal cost character for node 1 is either **g** or **t**. The minimal cost character for node 3 is **g**. The maximum parsimony approach would prefer the other tree, because it has a smaller cost (0.8).

Exploring the Space of Trees

- we've considered how to find the minimum number of changes for a given tree topology
- need some search procedure for exploring the space of tree topologies

Heuristic Method: Nearest Neighbor Interchange

- for any internal edge in a tree, there are 3 ways the four subtrees can be grouped
- nearest neighbor interchanges move from one grouping to another

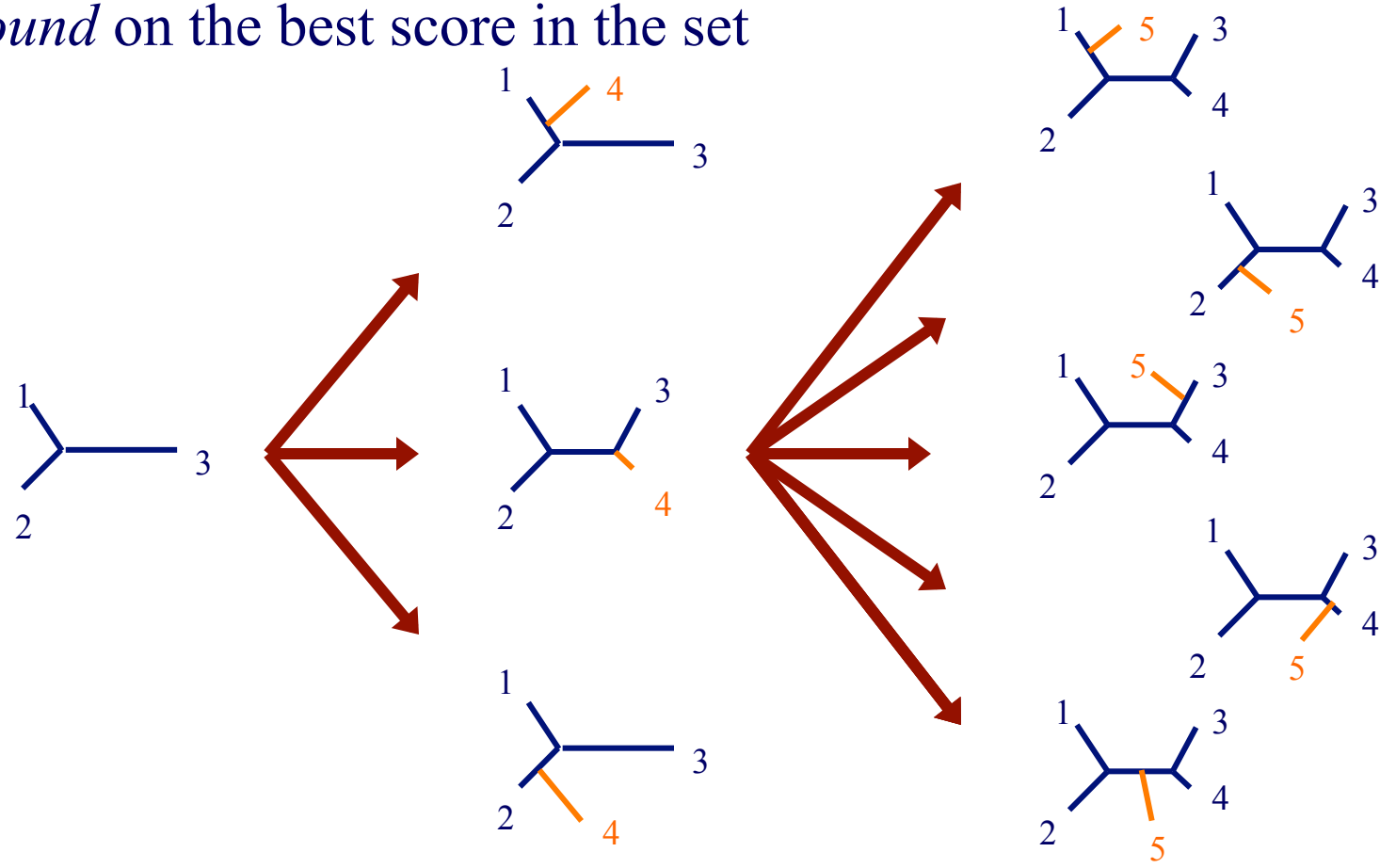


Heuristic Method: Hill-climbing with Nearest Neighbor Interchange

given: set of leaves *L*
create an initial tree *t* incorporating all leaves in *L*
best-score = parsimony algorithm applied to *t*
repeat
    for each internal edge *e* in *t*
        for each nearest neighbor interchange
            t' ← tree with interchange applied to edge *e* in *t*
            score = parsimony algorithm applied to *t'*
            if *score* < *best-score*
                best-score = *score*
                best-tree = *t'*
    t = *best-tree*
until stopping criteria met

Exact Method: Branch and Bound

- each partial tree represents a set of complete trees
- the parsimony score on a partial tree provides a *lower bound* on the best score in the set



- search by repeatedly selecting the partial tree with the lowest lower bound

Exact Method: Branch and Bound

given: set of leaves L

initialize Q with a partial tree with 3 leaves from L

repeat

    $t \leftarrow$ tree in Q with lowest lower bound

    if t has incorporated all leaves in L

        return t

    else

        create new trees by adding next leaf from L to each branch of t

        compute lower bound for each tree

        put trees in Q sorted by lower bound

Branch and Bound (Alternate Version)

given: set of leaves L

use heuristic method to grow initial tree t'

initialize Q with a partial tree with 3 leaves from L

repeat

    $t \leftarrow$ tree in Q with lowest lower bound

    if t has incorporated all leaves in L

        return t

    else

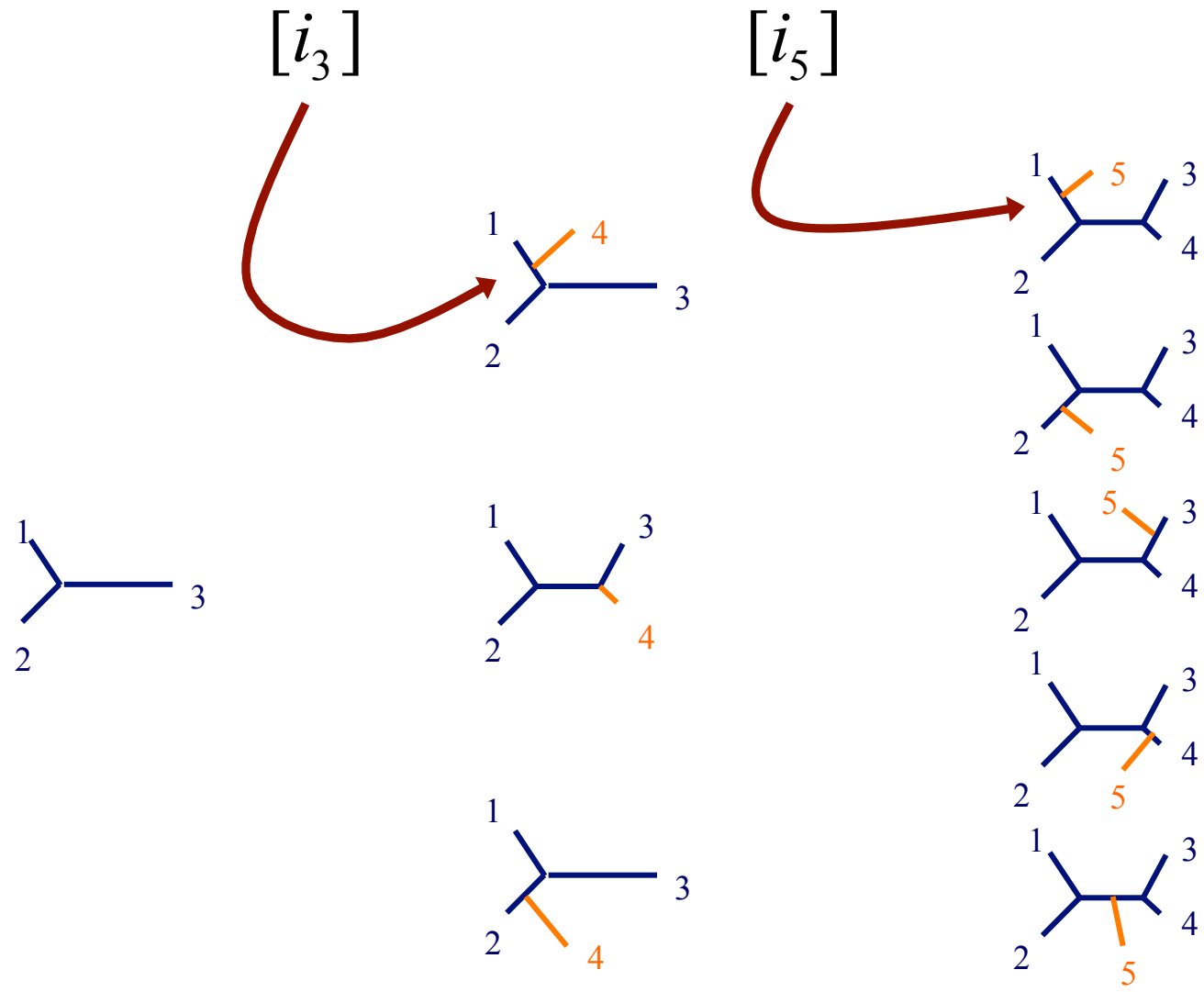
        create new trees by adding next leaf from L to each branch of t

        for each new tree n

            if lower-bound(n) < score(t')

            put n in Q sorted by lower bound

Implementing Branch and Bound (Second Alternate Version)



Implementing Branch and Bound (Second Alternate Version)

- for n sequences, maintain an array of counters

$$[i_3][i_5][i_7]\ldots[i_{2n-5}]$$

  where i_k takes on values $0 \ldots k$
- a complete tree is represented by an assignment of all i_k to non-zero values
- i_k indicates, for a partial tree with k edges, on which edge to add a branch for the next sequence
- $i_k = 0$ indicates a partial tree

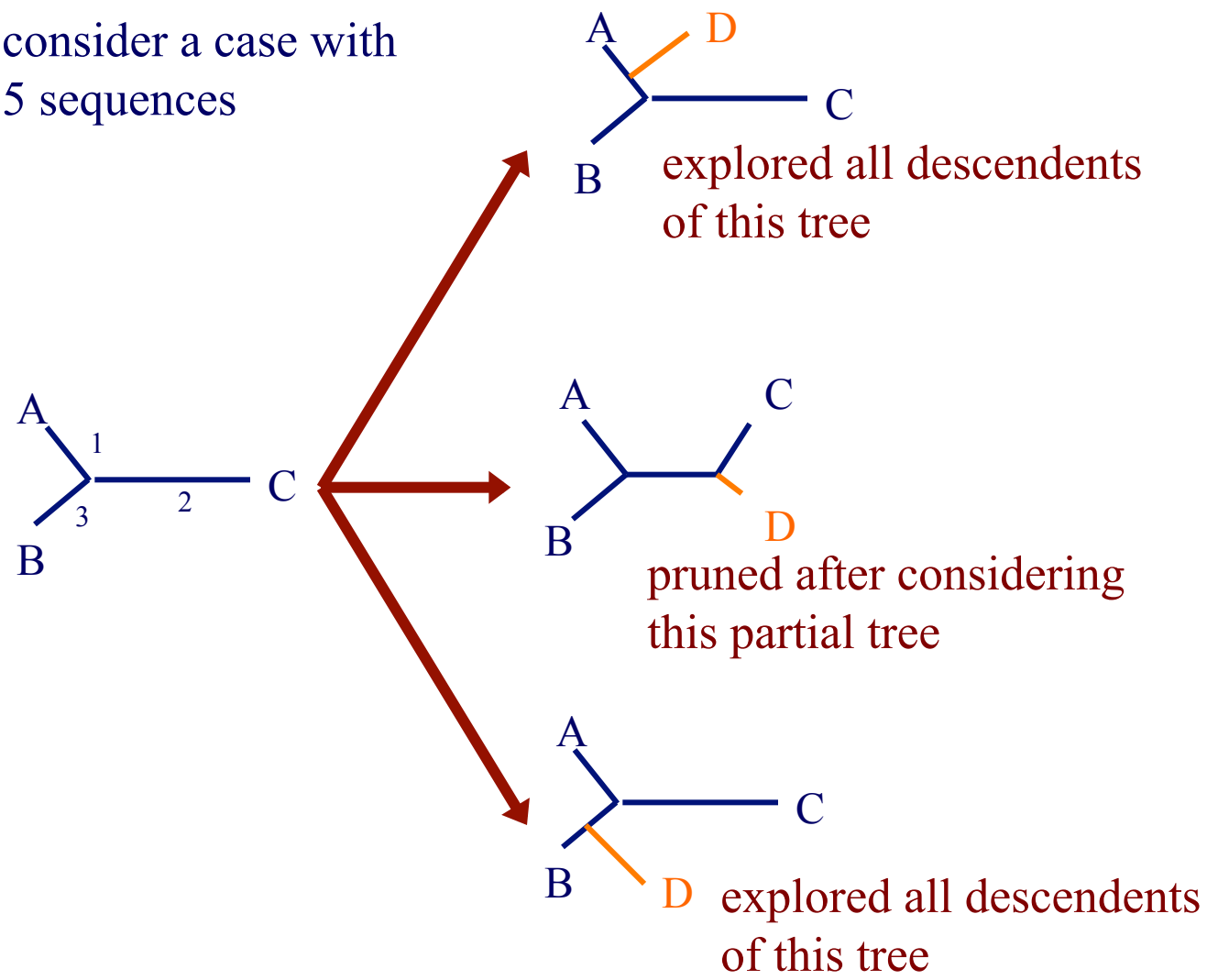
Implementing Branch and Bound

- to search space, roll counters through their allowable numbers (somewhat) like an odometer
  - rightmost counter moves fastest
  - whenever a counter is 0, all counters to the right of it must be 0 also
  - test cost of (partial) tree at each tick of odometer
  - have odometer skip when pruning occurs

$$[i_3][i_5][i_7]\ldots[i_{2n-5}]$$

Implementing Branch and Bound

consider a case with
5 sequences



counters went:

$[i_3]$	$[i_5]$
1	0
1	1
1	2
1	3
1	4
1	5
2	0
3	0
3	1
3	2
3	3
3	4
3	5

Comments on Branch and Bound

- it is a *complete* search method
  - guaranteed to find optimal solution
- may be much more efficient than exhaustive search
- in the worst case, it is no better
- efficiency depends
  - the tightness of the lower bound
  - the quality of the initial tree

Rooted or Unrooted Trees for Parsimony?

- we described parsimony calculations in terms of rooted trees
- but we described the search procedures in terms of unrooted trees

- *unweighted parsimony*: minimum cost is independent of where root is located
- *weighted parsimony*: minimum cost is independent of root if substitution cost is a metric (refer back to definition of metric from distance-based methods)

Comments on Tree Inference

- search space may be large, but
  - can find the optimal tree efficiently in some cases
  - heuristic methods can be applied
- difficult to evaluate inferred phylogenies: ground truth not usually known
  - can look at agreement across different sources of evidence
  - can look at repeatability across subsamples of the data
- some newer methods use data based on linear order of orthologous genes along chromosome
- phylogenies for bacteria, viruses not so straightforward because of *lateral transfer* of genetic material