BMEG3102 Bioinformatics

Lecture 5. Mutation Models and Molecular Phylogenetics (2/2)

Qi Dou

Email: qidou@cuhk.edu.hk

Office: Room 1014, 10/F, SHB

BMEG3102 Bioinformatics

The Chinese University of Hong Kong

Lecture outline

1. Phylogenetic tree reconstruction

- Problem definition

2. Distance-based methods

- UPGMA
- Neighbor-joining

3. Sequence-based methods

- Maximum parsimony
- Maximum likelihood

Part 1

Phylogenetic Tree Reconstruction

Phylogenetic tree reconstruction

- General problem:
  - Given a set of DNA/protein sequences
  - Find a phylogenetic tree such that it likely corresponds to the actual historical evolutionary events, involving:
    - Order of separation events (how the nodes are connected)
    - Ancestral sequences (what sequences the internal nodes have)
    - Branch lengths (how much time it has been since the separation)

    There are various ways to evaluate how likely a tree is correct.

    We will study them in this lecture

  - "Re"-construction: The tree was defined by history. We only try to reconstruct it from the observed sequences

What sequence to use?

- If we are studying a gene
  - DNA/protein sequence of the gene

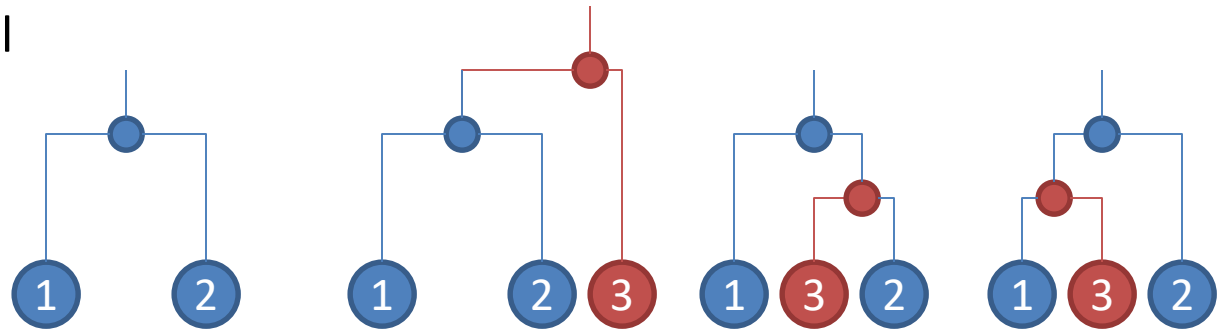
- If we want to know the relationship between different species
  - Whole genome (may not be feasible)
  - Some genes that are essential and single-copied
    - Ribosomal RNA

Complexity of problem

- Finding the “best” tree is a hard problem
- How many tree topologies (i.e., ignore branch lengths and left-right order) are there for a set of k sequences?

Complexity of problem

- Finding the “best” tree is a hard problem
- How many tree topologies (i.e., ignore branch lengths and left-right order) are there for a set of k sequences?
- For rooted trees:
  - k=2: 1 possible tree topology
  - k=3: 3 possible branches to add #3
  - k=4: 5 possible branches to add #4, and so on
  - Therefore, number of tree topologies is $1 \times 3 \times 5 \times \ldots \times (2k-3)$
    - Exponential

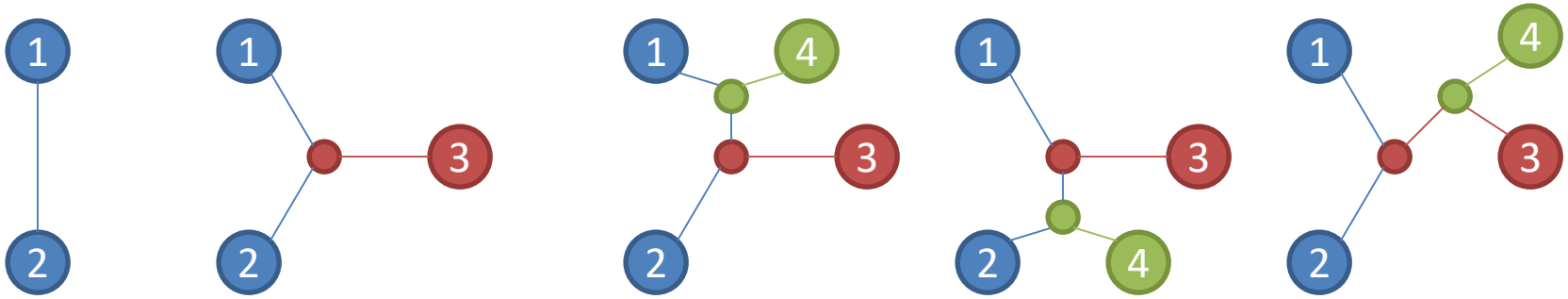


k	Num. of rooted tree topologies
2	1
3	3
4	15
5	105
6	945
7	10,395
8	135,135
9	2,027,025
10	34,459,425
11	654,729,075
12	13,749,310,575
13	316,234,143,225
14	7,905,853,580,625
15	213,458,046,676,875
16	6,190,283,353,629,370
17	191,898,783,962,511,000
18	6,332,659,870,762,850,000
19	221,643,095,476,700,000,000
20	8,200,794,532,637,890,000,000

Complexity of problem

- Similarly, for unrooted trees,
  - k=2: 1 possible tree topology
  - k=3: 1 possible branch to add #3
  - k=4: 3 possible branches to add #4
  - k=5: 5 possible branches to add #5
  - There number of tree topologies is $1 \times 3 \times 5 \times \ldots \times (2k-5)$

k	Num. of rooted tree topologies	Num. of unrooted tree topologies
2	1	1
3	3	1
4	15	3
5	105	15
6	945	105
7	10,395	945
8	135,135	10,395
9	2,027,025	135,135
10	34,459,425	2,027,025
11	654,729,075	34,459,425
12	13,749,310,575	654,729,075
13	316,234,143,225	13,749,310,575
14	7,905,853,580,625	316,234,143,225
15	213,458,046,676,875	7,905,853,580,625
16	6,190,283,353,629,370	213,458,046,676,875
17	191,898,783,962,511,000	6,190,283,353,629,370
18	6,332,659,870,762,850,000	191,898,783,962,511,000
19	221,643,095,476,700,000,000	6,332,659,870,762,850,000
20	8,200,794,532,637,890,000,000	221,643,095,476,700,000,000



Solving the problem: Ideas

- What do you do when you encounter a computationally hard problem?
  - Define an easier version of the problem
    - By making certain assumptions
  - Design smart algorithms/data structures to avoid redundant calculations
  - Use heuristics to solve it, not necessarily getting the optimal solution

Phylogenetic tree reconstruction methods

- Two main types of methods:
  - Sequence-based: need the sequences
    - Parsimony methods (easier problems, smart algorithms)
    - Probabilistic methods (easier problems, smart algorithms)
      - Maximum likelihood
      - Bayesian
    - ...
  - Distance-based: only depends on the distances between the sequences
    - UPGMA (heuristics)
    - Neighbor joining (heuristics)
    - ...
  - We will study some of these algorithms

Lecture scope

- What you need to know:
  - Basic concepts behind each type of algorithms
  - How to use the methods to solve simple problems in phylogenetic tree reconstruction
    - Except the maximum likelihood method

- What you are NOT required to know:
  - Details of the maximum likelihood method
  - Proof of the correctness of the algorithms
  - How to implement the methods in a programming language

Part 2a

Distance-based Methods: UPGMA

Motivation

- In the sequence-based algorithms, the exact sequences are used when reconstructing the phylogenetic trees

- In a distance-based method, only the pairwise distances between the sequences are considered
  - Good if the sequences are long, and we care only about the tree structure but not the ancestral sequences
  - The distances can be computed by methods based on sequence alignment
  - Once the pairwise distances have been computed, the original sequences will not be used

UPGMA

- Unweighted Pair Group Method with Arithmetic Mean
- Algorithm:
  1. Compute the distance between each pair of sequences
  2. Treat each sequence as a cluster by itself
  3. Merge the two closest clusters. The distance between two clusters is the average distance between all their sequences (except that $d(C_i, C_i)=0$):

$$d(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{r \in C_i, s \in C_j} d(r, s)$$

  (Notice that $d(r, s)$ is the distance between r and s in the input distance matrix)

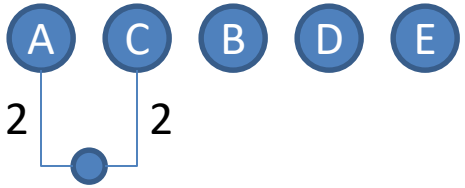
  4. Repeat 2 and 3 until only one cluster remains

Example

	A	B	C	D	E
A	0	8	4	6	8
B	8	0	8	8	4
C	4	8	0	6	8
D	6	8	6	0	8
E	8	4	8	8	0

{A},{C}

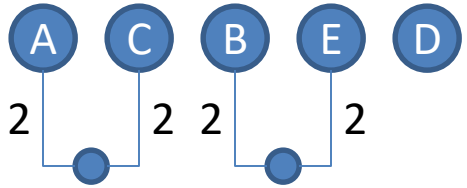
	A,C	B	D	E
A,C	0	8	6	8
B	8	0	8	4
D	6	8	0	8
E	8	4	8	0



Note: Here node labels are sequence names, not the actual characters/bases

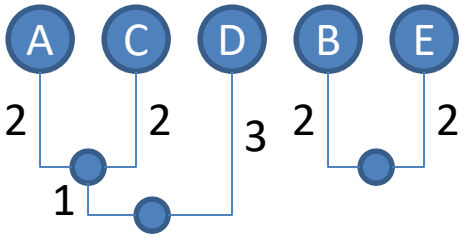
{B},{E}

	A,C	B,E	D
A,C	0	8	6
B,E	8	0	8
D	6	8	0



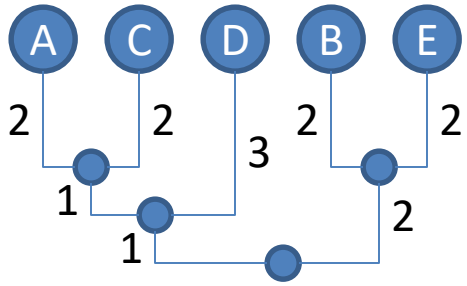
{A,C}, {D}

	A,C,D	B,E
A,C,D	0	8
B,E	8	0



{A,C,D}, {B,E}

	A,B,C,D,E
A,B,C,D,E	0



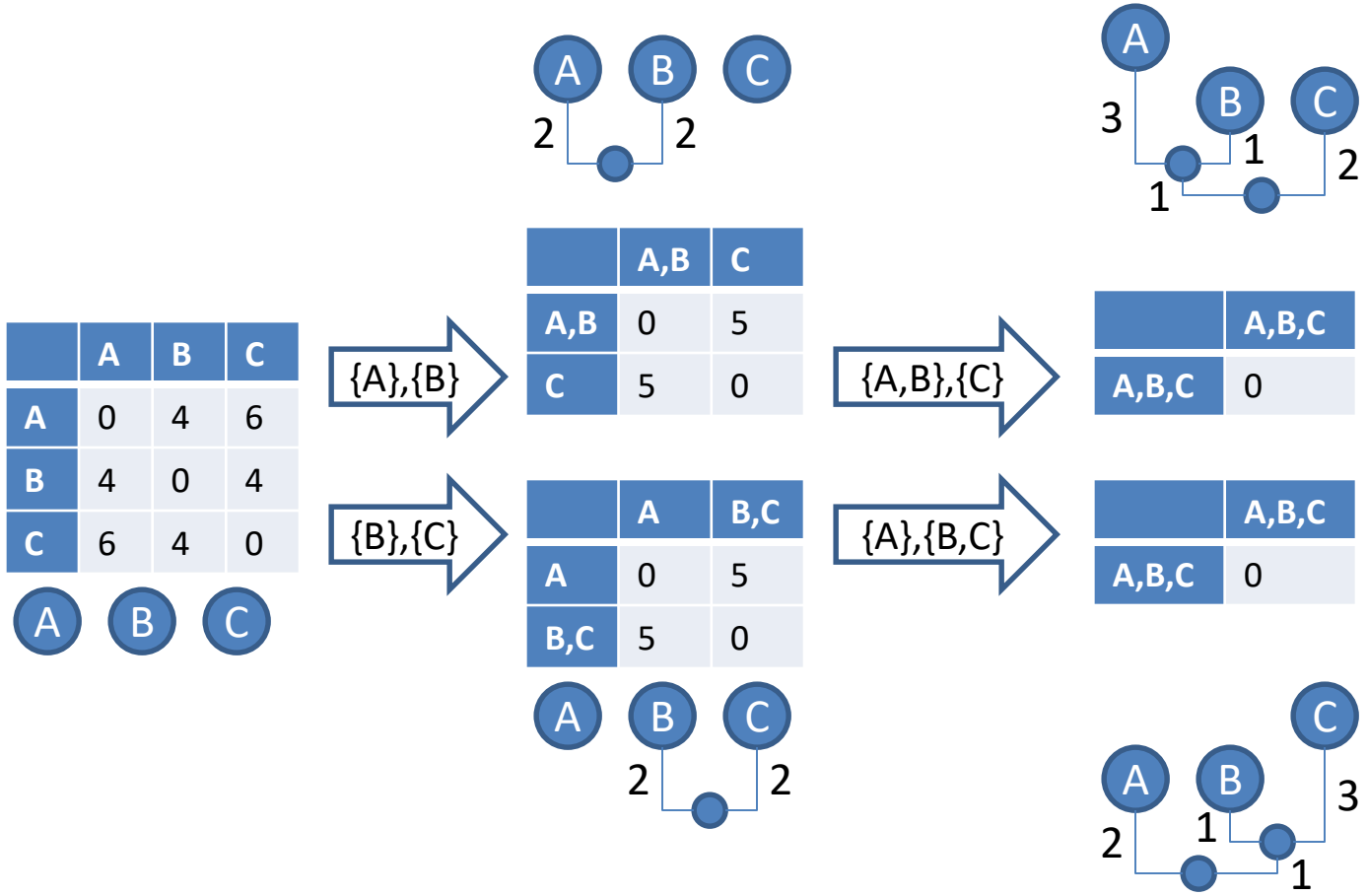
Note: In this case,

- The tree is unique
- Sum of branch lengths between two sequences equals their input distance
- All leaves are on the same horizontal line

Do we always have these properties?

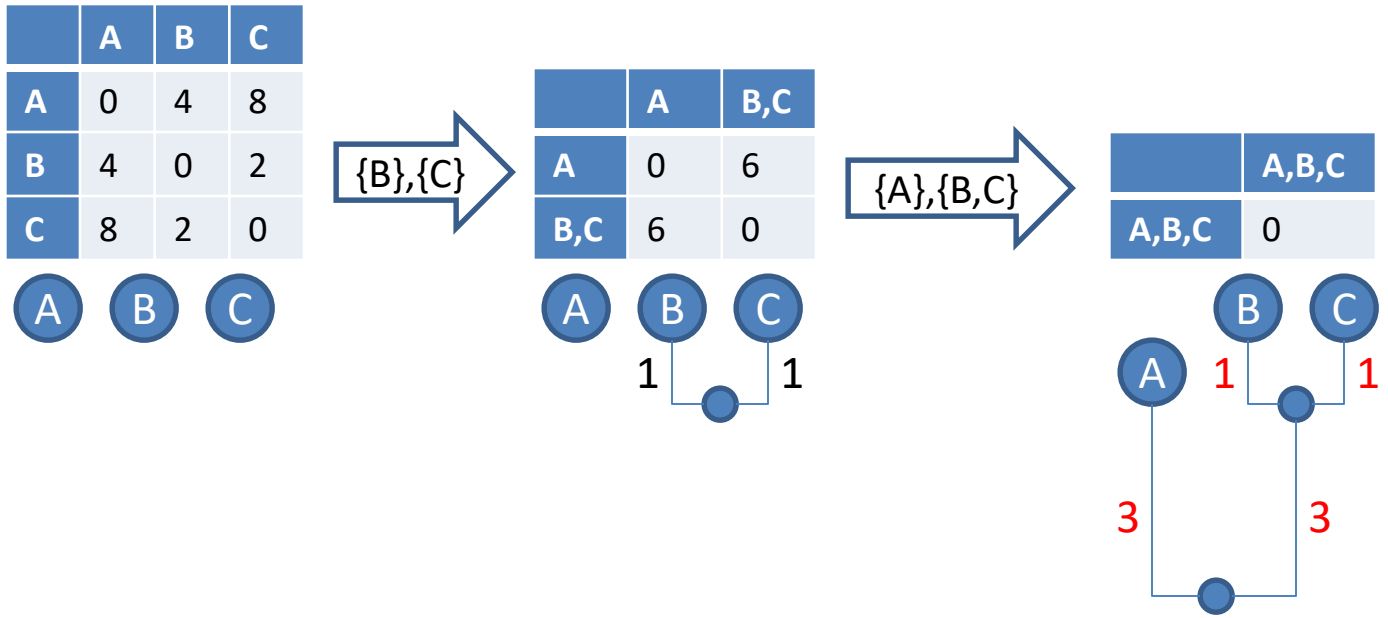
Uniqueness

- Not always unique, also not always possible to put all leaf nodes on a line:



Branch lengths

- Not always possible to assign branch lengths according to distances:



Here the branch lengths only reflect the cluster distances, not the sequence distances

Ultrametric distances [Optional]

- Additive tree: A tree where the length between two nodes is the total length of the branches between them
  - For example, when the length represents the number of observed + unobserved substitutions
- Desirable properties of the input distance matrix:
  - i. $d(x, y) \geq 0$
  - ii. $d(x, y) = 0$ if $x = y$
  - iii. $d(x, y) = d(y, x)$
  - iv. $d(x, y) + d(y, z) \geq d(x, z)$
  - v. $d(x, y) \leq \max\{d(x, z), d(y, z)\}$
- We can get an additive tree **and** with all leaf nodes on the same line (i.e., an ultrametric tree) if i, ii, iii, iv and v are true

	A	B	C	D	E
A	0	8	4	6	8
B	8	0	8	8	4
C	4	8	0	6	8
D	6	8	6	0	8
E	8	4	8	8	0

True: i, ii, iii, iv and v

	A	B	C
A	0	4	6
B	4	0	4
C	6	4	0

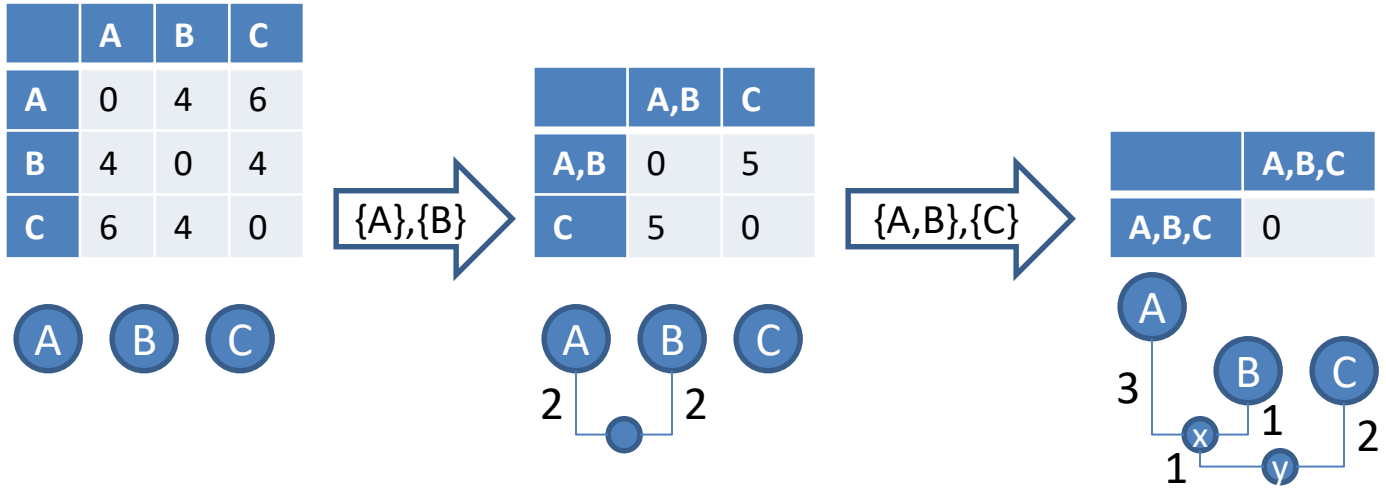
True: i, ii, iii and iv

	A	B	C
A	0	4	8
B	4	0	2
C	8	2	0

True: i, ii and iii

Additive trees [Optional]

- When a tree is additive, the branch lengths can be determined by solving simultaneous equations



- For non-additive trees, we will learn how to assign "reasonable" distances by neighbor joining

d(A,x) + d(B,x) = d(A,B) = 4 (1)
d(A,x) + d(x,y) + d(C,y) = d(A,C) = 6 (2)
d(B,x) + d(x,y) + d(C,y) = d(B,C) = 4 (3)
[(1) – (2) + (3)] / 2:
d(B,x) = 1
d(A,x) = 3
d(x,y) + d(C,y) = 3 [e.g., d(x,y) = 1, d(C,y) = 2]

Part 2b

Distance-based Methods: Neighbor Joining

Neighbor joining

- In UPGMA, each time we merge the two closest clusters according to their distance (criterion #1):

$$d(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{r \in C_i, s \in C_j} d(r, s)$$

- Would be good to choose the pair that is also far away from other clusters (criterion #2), measured by:

$$u(C_i) = \sum_j d(C_i, C_j)$$

Neighbor joining

- In UPGMA, each time we merge the two closest clusters according to their distance (criterion #1):

$$d(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{r \in C_i, s \in C_j} d(r, s)$$

- Would be good to choose the pair that is also far away from other clusters (criterion #2), measured by:

$$u(C_i) = \sum_j d(C_i, C_j)$$

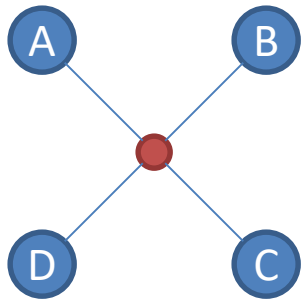
- In the Neighbor Joining algorithm, the two clusters to merge is the pair that minimizes $Q(i, j) = (r - 2)d(C_i, C_j) - u(C_i) - u(C_j)$, where r is the current number of clusters (and Q(i, i) ≡ 0 for all i)
  - The formula considers both criteria, while the (r-2) factor is to balance the relative weights of them

Neighbor joining

- The algorithm:
  1. Start with each sequence as a cluster. All of them are connected to a hub, forming a star.
  2. Find clusters i and j connected to the hub where Q(i, j) is minimum among all cluster pairs
  3. Insert a new internal node C_k
     - Connect it to C_i, C_j and the hub
     - Assign length $\frac{d(C_i, C_j)}{2} + \frac{u(C_i) - u(C_j)}{2(r-2)}$ to the edge C_iC_k (r is the number of clusters before the merge)
     - Assign length $\frac{d(C_i, C_j)}{2} + \frac{u(C_j) - u(C_i)}{2(r-2)}$ to the edge C_jC_k
     - For each node C_l, $d(C_k, C_l) = [d(C_i, C_l) + d(C_j, C_l) - d(C_i, C_j)] / 2$
     (Notice that all $d(C_x, C_y)$ values are from the previous step.)
  4. Repeat 2 and 3 until all branch lengths are assigned
     - The final result will be an unrooted tree

Example

$$Q(i,j) = (r-2)d(C_i,C_j) - u(C_i) - u(C_j)$$



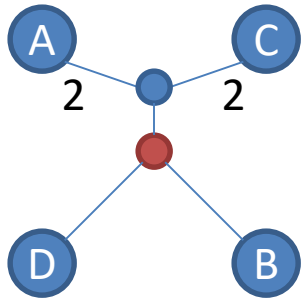
d	A	B	C	D
A	0	8	4	6
B	8	0	8	8
C	4	8	0	6
D	6	8	6	0

u	
A	18
B	24
C	18
D	20

Q	A	B	C	D
A	0	-26	-28	-26
B	-26	0	-26	-28
C	-28	-26	0	-26
D	-26	-28	-26	0

Distance between A and the new node: $d(A,C)/2 + [u(A) - u(C)] / [2(r-2)] = 4/2 + (18-18) / [2(2)] = 2$

{A}, {C}



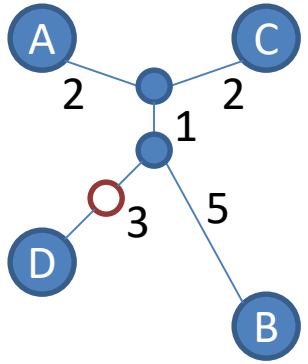
d	A,C	B	D
A,C	0	6	4
B	6	0	8
D	4	8	0

u	
A,C	10
B	14
D	12

Q	A,C	B	D
A,C	0	-18	-18
B	-18	0	-18
D	-18	-18	0

$$d(C_k, C_l) = [d(C_i, C_l) + d(C_j, C_l) - d(C_i, C_j)] / 2$$

{A,C}, {B}



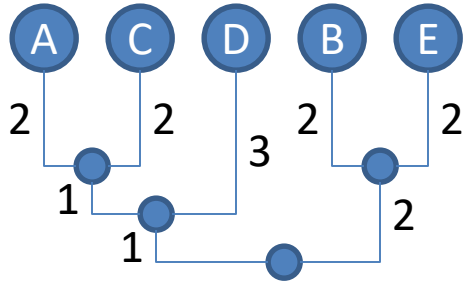
d	A,B,C	D
A,B,C	0	3
D	3	0

u	
A,B,C	3
D	3

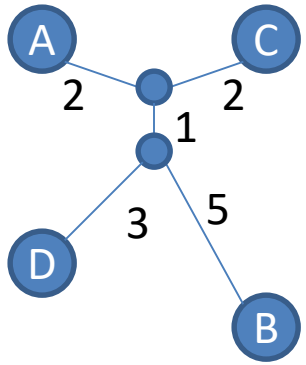
In the last step, we simply remove the hub and write down the distance

Comparing the results

- UPGMA: (with one more node, E)



- Neighbor Joining:



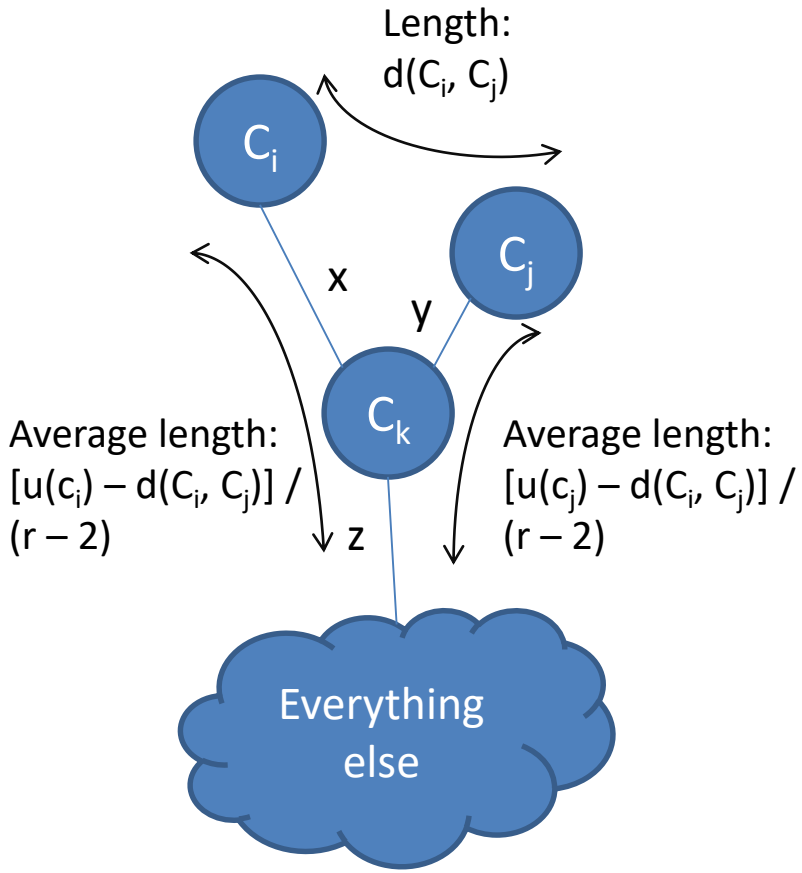
Reason for the branch length assignment

- If distances are additive:
  - $x + z = [u(c_i) - d(C_i, C_j)] / (r - 2)$
  - $y + z = [u(c_j) - d(C_i, C_j)] / (r - 2)$
  - $x + y = d(C_i, C_j)$
  - $\Rightarrow [(x + z) - (y + z) + (x + y)] / 2 =$

$$x = \frac{d(C_i, C_j)}{2} + \frac{u(C_i) - u(C_j)}{2(r-2)}$$

  - and $[(y + z) - (x + z) + (x + y)] / 2 =$

$$y = \frac{d(C_i, C_j)}{2} + \frac{u(C_j) - u(C_i)}{2(r-2)}$$



Reason for the branch length assignment

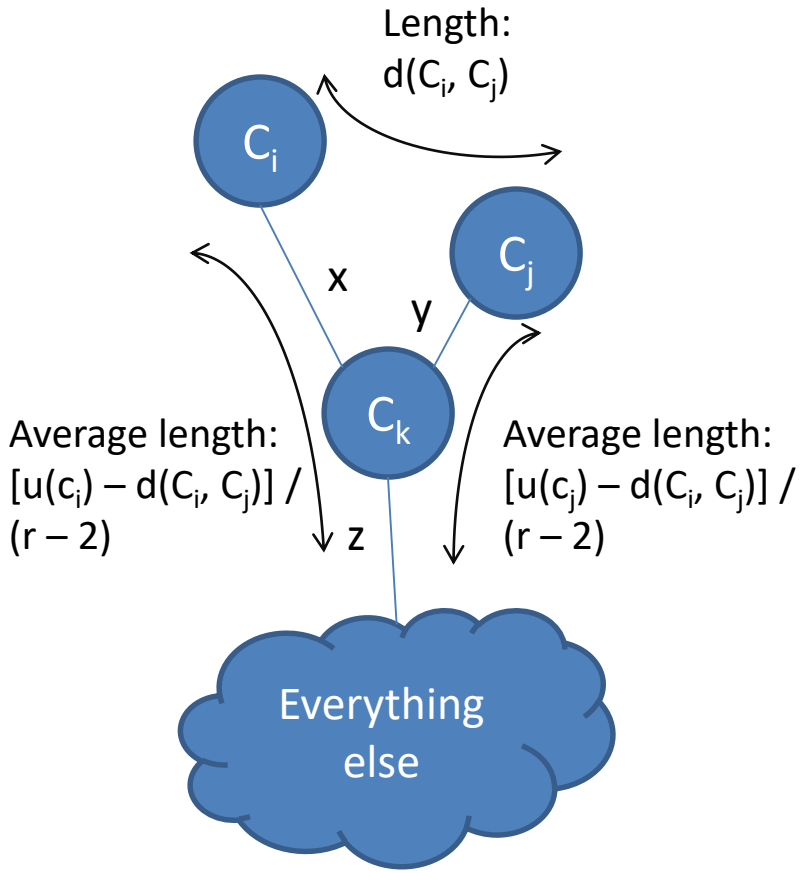
- If distances are additive:
  - $x + z = [u(c_i) - d(C_i, C_j)] / (r - 2)$
  - $y + z = [u(c_j) - d(C_i, C_j)] / (r - 2)$
  - $x + y = d(C_i, C_j)$
  - $\Rightarrow [(x + z) - (y + z) + (x + y)] / 2 =$

    $$x = \frac{d(C_i, C_j)}{2} + \frac{u(C_i) - u(C_j)}{2(r-2)}$$

  - and $[(y + z) - (x + z) + (x + y)] / 2 =$

    $$y = \frac{d(C_i, C_j)}{2} + \frac{u(C_j) - u(C_i)}{2(r-2)}$$

- If distances are not additive, the assigned distances are still usually reasonable
- $d(C_k, C_l) = [(d(C_i, C_l) + d(C_j, C_l) - d(C_i, C_j)] / 2$



Rooting an unrooted tree

- How to find the root of an unrooted tree?
  - Usually by using an “out group”, something that should be separated first
  - There are some other methods

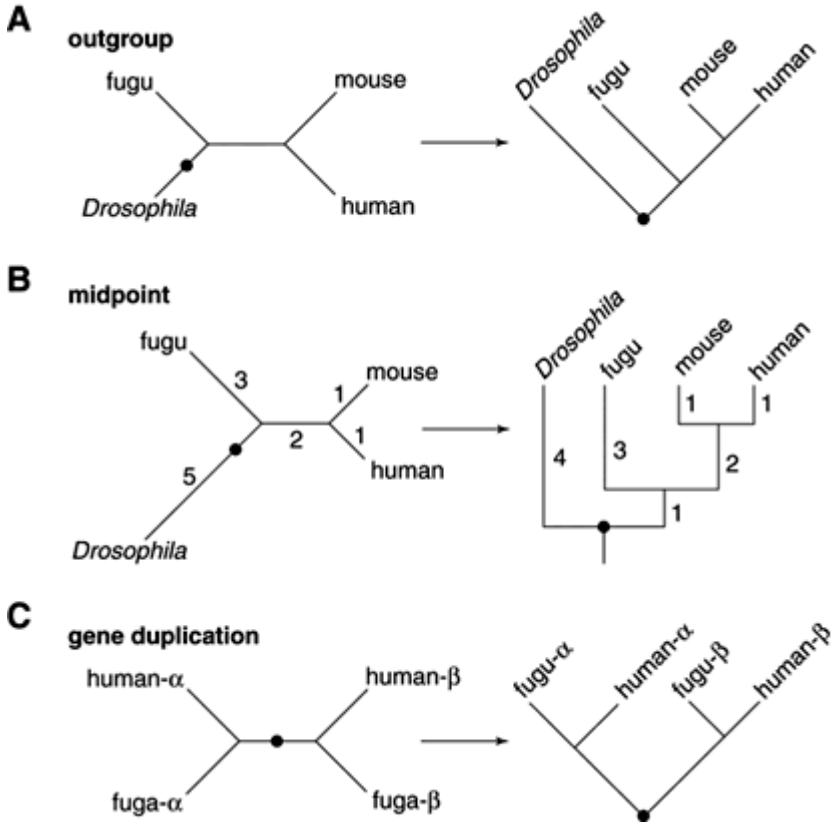


Image credit: Wikipedia, http://blog.ohinternet.com/wp-content/uploads/2011/03/fugu.jpg , http://www.currentprotocols.com/protocol/bi0601

Part 3a

Sequence-based Methods: Maximum Parsimony

Maximum parsimony

- Assumption: A tree is likely to be true if it involves few mutations
- Rationale:
  - Mutations are rare
  - “Occam’s razor”: The simplest explanation is likely the correct one

Maximum parsimony

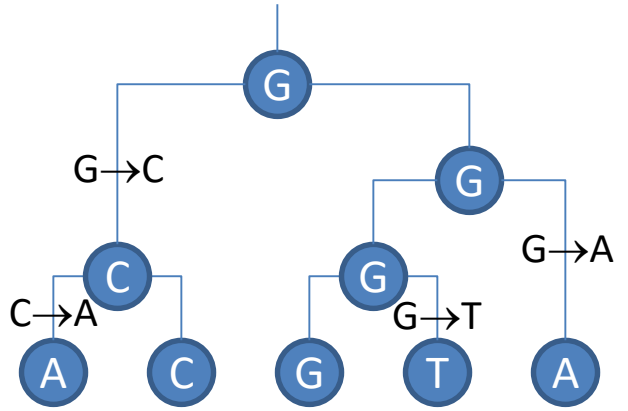
- Assumption: A tree is likely to be true if it involves few mutations
- Rationale:
  - Mutations are rare
  - “Occam’s razor”: The simplest explanation is likely the correct one
- “Large parsimony” problem:
  - Given a set of sequences
  - Find a rooted tree topology of the sequences and the ancestral sequences of the tree
  - Such that the total number of mutations along the branches is minimized
  - NP hard: Currently no polynomial time algorithm is known

Maximum parsimony

- Assumption: A tree is likely to be true if it involves few mutations
- Rationale:
  - Mutations are rare
  - "Occam's razor": The simplest explanation is likely the correct one
- "Large parsimony" problem:
  - Given a set of sequences
  - Find a rooted tree topology of the sequences and the ancestral sequences of the tree
  - Such that the total number of mutations along the branches is minimized
  - NP hard: Currently no polynomial time algorithm is known
- "Small parsimony" problem:
  - Given a set of sequences and a rooted tree topology of the sequences
  - Find the ancestral sequences
  - Such that the total number of mutations along the branches is minimized
- We will focus on the small parsimony problem

Small parsimony example

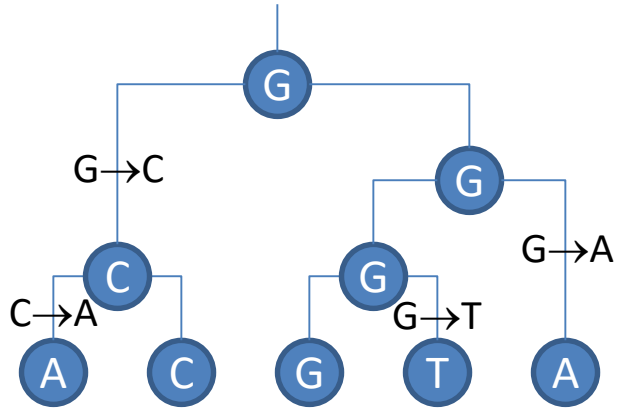
- We will consider one single site
  - By assuming that sites are independent, we only need an algorithm for one site
  - Will show an example with more sites later

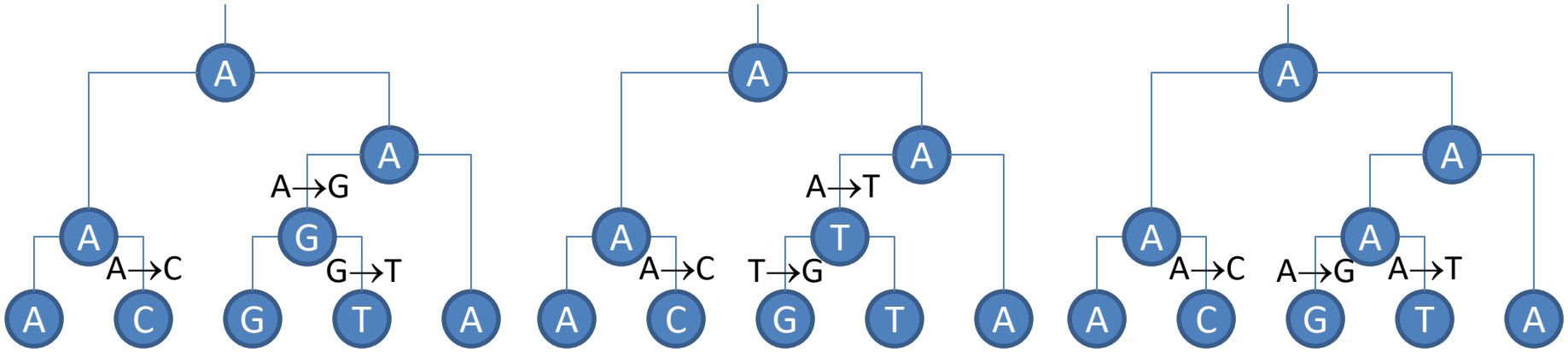


Small parsimony example

- We will consider one single site
  - By assuming that sites are independent, we only need an algorithm for one site
  - Will show an example with more sites later
- In the upper tree on the right, the number of mutations is 4
  - Is it the minimum (i.e., most parsimonious solution)?
  - For this tree topology, the minimum number of mutations is 3.

    There are three sets of ancestral states that result in this number of mutations, shown in the three trees below





Small parsimony problem

- How to assign ancestral states so that the total number of mutations is minimized?

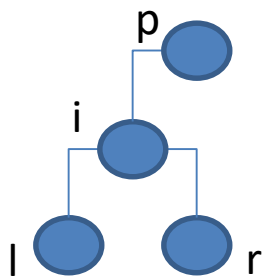
- Ideas: For a given node,
  - If both children have the same state, probably good to adopt the state
  - If the two children have different states, probably good to adopt one of them
  - Delay the decision of the exact choice until the parent has also expressed a preference

The algorithm: simple version

- Fitch’s algorithm: If you only need some solutions

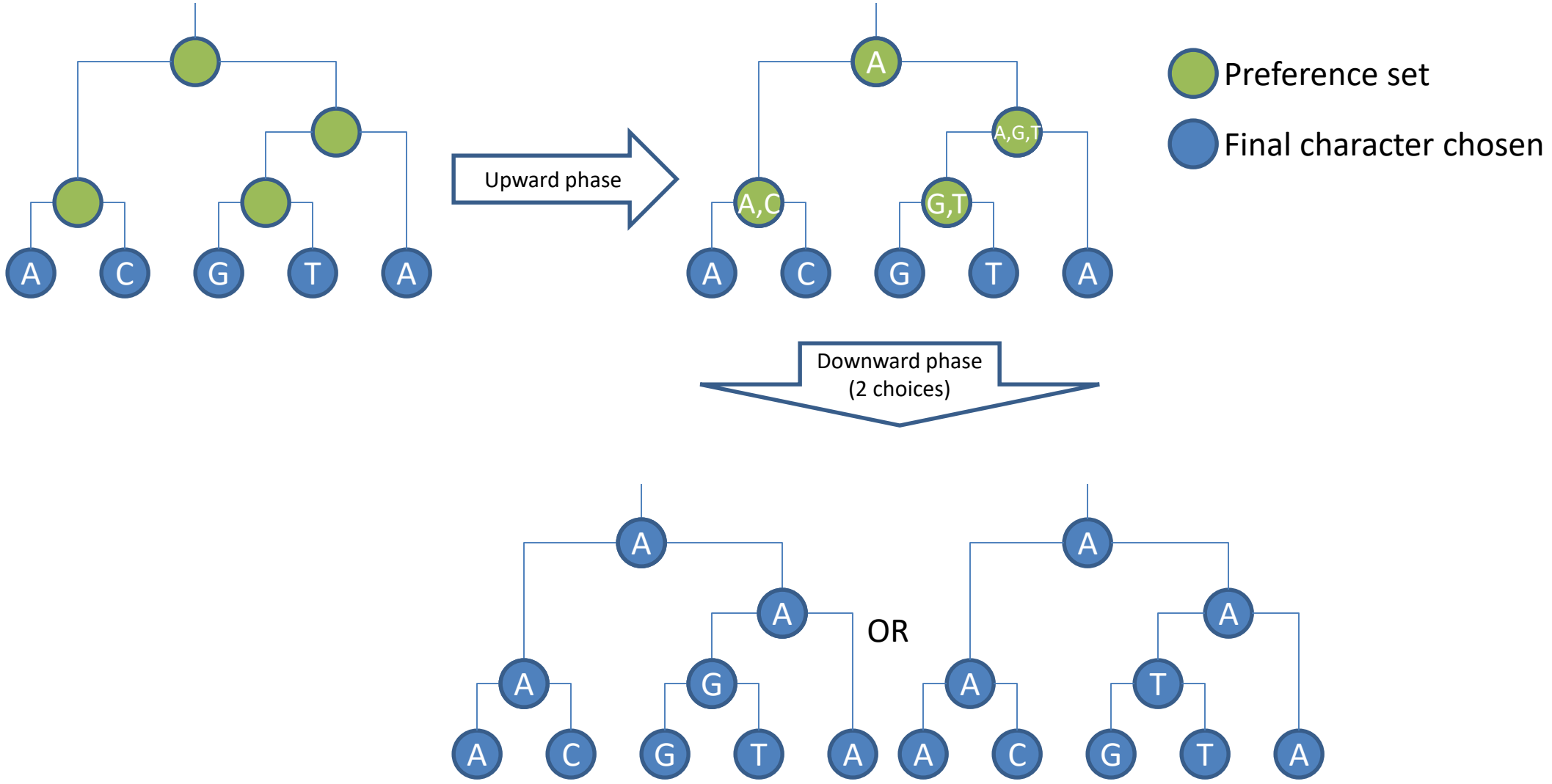
The algorithm: simple version

- Fitch's algorithm: If you only need some solutions
  - For each internal node i with parent p and children l and r, we will determine its preference set S_i and its final character C_i that would minimize the total number of mutations
  - Steps:



1. For each leaf node i, set S_i to the character of the sequence
2. Upward phase: For each internal node i,
   if ($S_l \cap S_r$)={} // l and r do not agree: take both sets
   $S_i := S_l \cup S_r$
   else // l and r agree on something: take the agreed part
   $S_i := S_l \cap S_r$
3. Downward phase: First pick any C_{root} from S_{root}. Then for each other internal node i,
   if $C_p \in S_i$ // p agrees with i on something: take it
   $C_i := C_p$
   else // p disagrees with i: use i's own preferences
   C_i := choose one from S_i

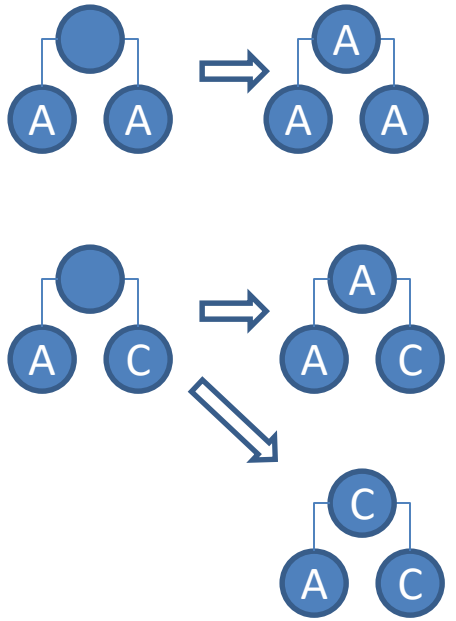
An example


A
C
G
T
A
Upward phase
A
A,G,T
A,C
G,T
A
C
G
T
A
Preference set
Final character chosen
Downward phase
(2 choices)
A
A
A
G
A
C
G
T
A
OR
A
A
A
T
A
C
G
T
A

Why does it work? [Optional]

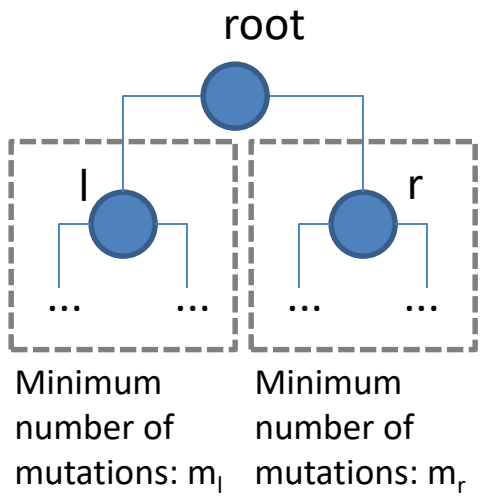
- Proof by induction
  - When there are two leaves, there are only two cases:
    - They have the same character
      - Actual minimum number of mutations: 0
      - The algorithm gives the same number
    - They have different characters
      - Actual minimum number of mutations: 1
      - The algorithm also gives the same number

  Therefore, the algorithm is optimal



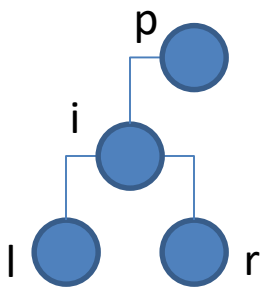
Why does it work? [Optional]

- Assume the algorithm is able to minimize the number of mutations for trees with k or fewer leaves
- Now for a tree with k+1 leaves,
  - It consists of a root connected to two sub-trees with roots l and r, both with k or fewer leaves
  - Two cases:
    - If $S_l \cap S_r \neq \{\}$, the algorithm gives a solution with $m_l + m_r$ mutations, which is optimal due to the induction hypothesis
    - If $S_l \cap S_r = \{\}$, the algorithm gives a solution with $m_l + m_r + 1$ mutations, which is also optimal since one extra mutation must be introduced between the root and one of its children



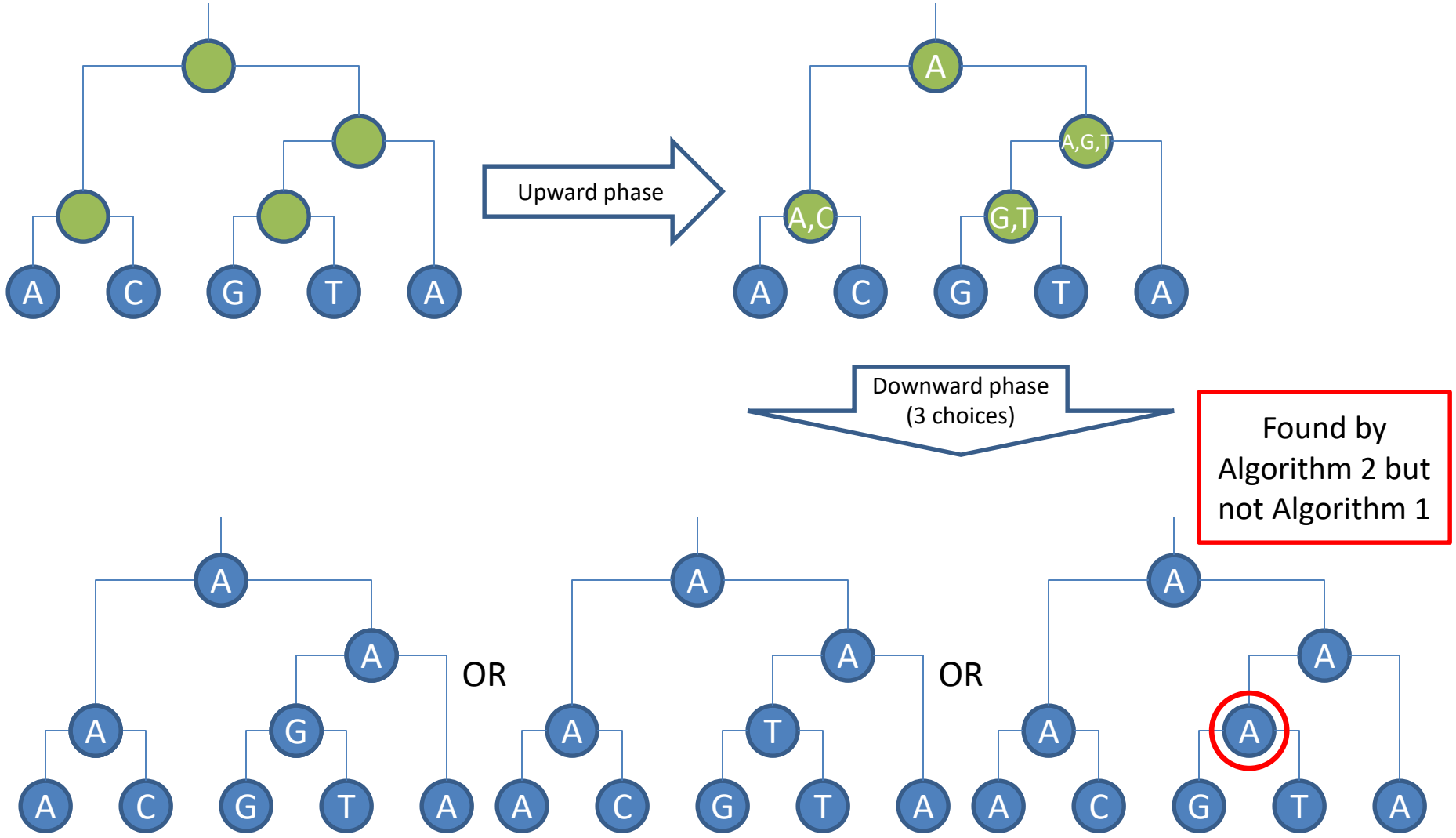
The algorithm: extended version

- If you need all solutions
  - Steps:
    1. For each leaf node i, set S_i to the character of the sequence
    2. Upward phase (same as before): For each internal node i,
       if ($S_l \cap S_r$)={} // l and r do not agree: take both sets
       $S_i := S_l \cup S_r$
       else // l and r agree on something: take it
       $S_i := S_l \cap S_r$
    3. Downward phase: First pick C_{root} from S_{root}. Then for each other internal node i **(different strategy -- majority vote): we will choose C_i from the characters that exist in the largest number of sets among {C_p}, S_l and S_r. Also, whenever there are multiple choices, we choose each in turn to enumerate all optimal solutions.**

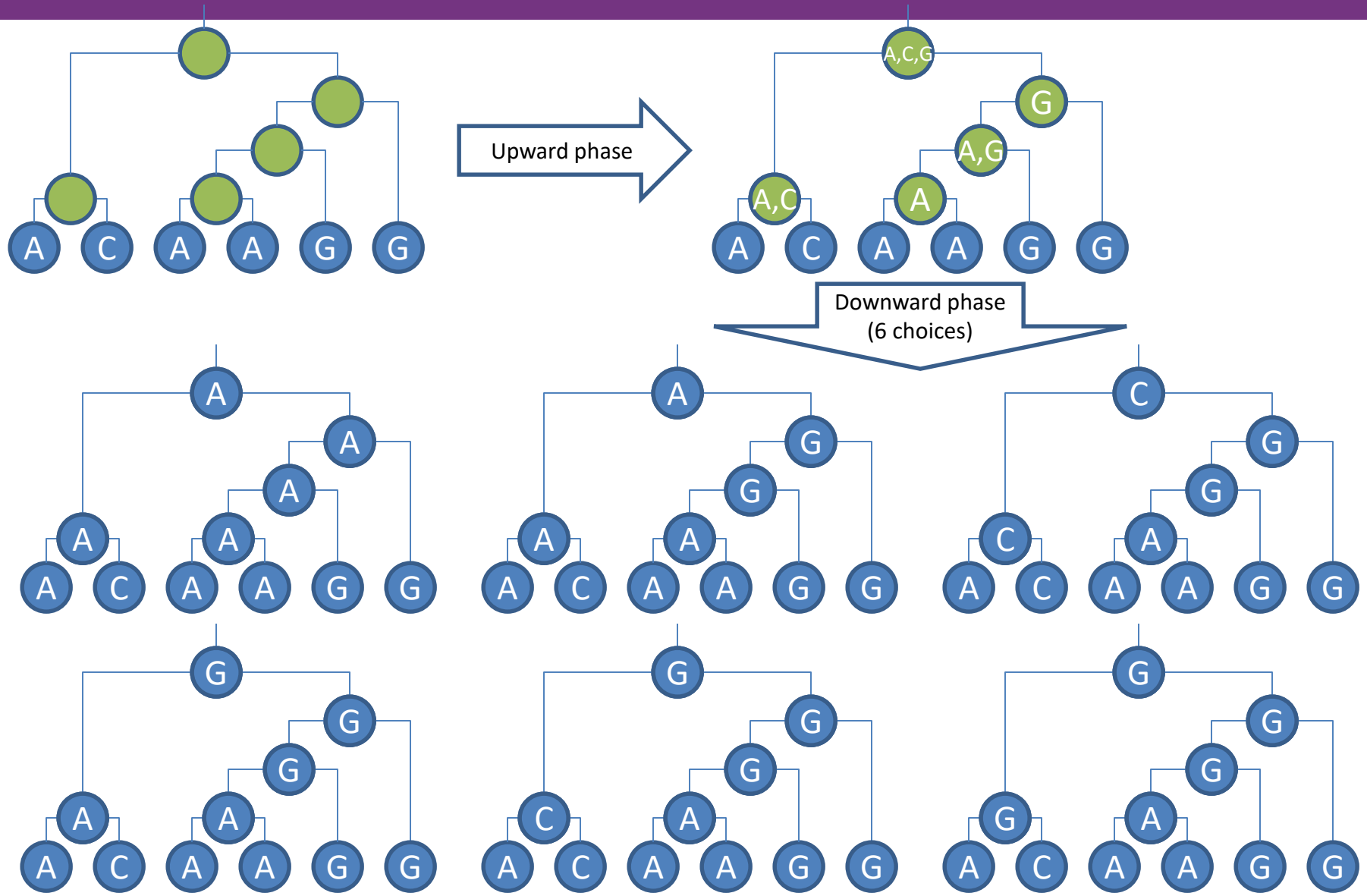


  - A special case of Sankoff's dynamic programming algorithm

Revisiting the same example



A more complex example



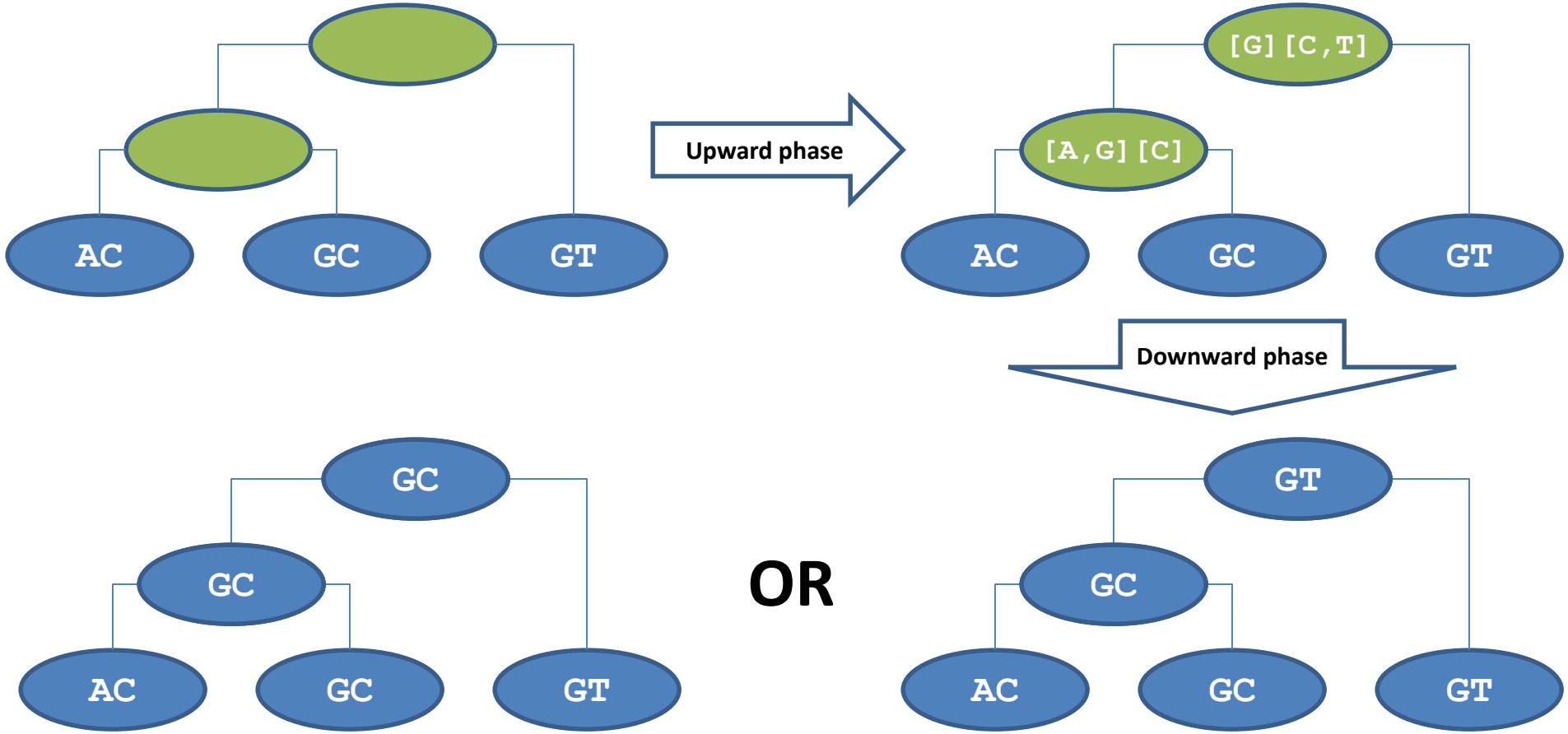
Multiple sites

- In a real situation, we need to deal with sequences that contain more than one site

- We simply apply the above algorithm to the different sites independently
  - As we assume that different sites mutate independently

Example

- Minimum: 1 substitution for position 1, 1 substitution for position 2
- Maximum parsimony: 2 trees that can achieve this minimum



Part 3b

Sequence-based Methods: Maximum Likelihood

Maximum likelihood

- Likelihood: Probability of producing the observed data by a model given the model parameters, $\Pr(X|\theta)$
  - X: Observed data
    - The input sequences, assumed aligned
    - Again, we consider one single site here. The likelihood for the whole sequences is the product of the likelihood of individual sites since they are assumed independent
  - θ: Model parameters (see next page)

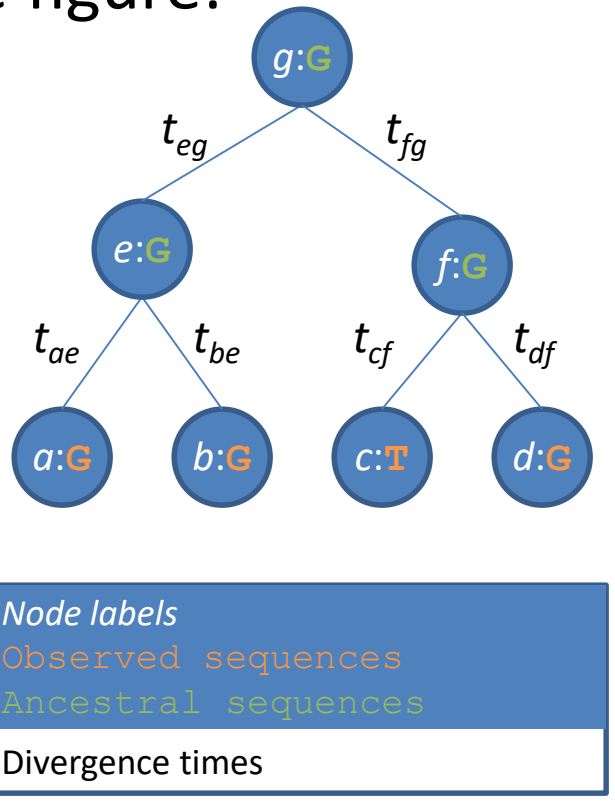
- Maximum likelihood: Find value of θ such that $\Pr(X|\theta)$ is maximized

Model parameters

- There are different possibilities
  - In all cases, *X* is the input sequences
- Big likelihood problem
  - θ: tree topology, mutation rates and divergence times
  - Very difficult
- Small likelihood problem
  - Tree topology is given
  - θ: mutation rates and divergence times
  - There are effective heuristic solutions that usually (but not always) produce optimal results

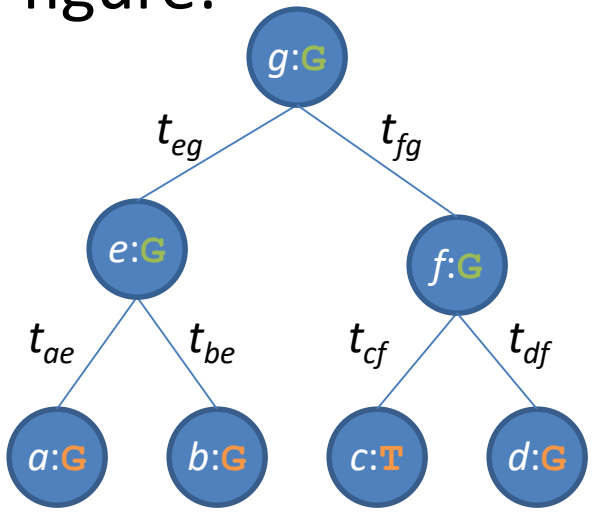
Computing likelihood

- Suppose we are given the followings, as shown in the figure:
  - Tree topology
  - Observed data, X = {a:G, b:G, c:T, d:G}
  - Ancestral sequences
  - Parameters, θ = {<mutation rates>, t_{ae}, t_{be}, t_{cf}, t_{df}, t_{eg}, t_{fg}}



Computing likelihood

- Suppose we are given the followings, as shown in the figure:
  - Tree topology
  - Observed data, X = {a:G, b:G, c:T, d:G}
  - Ancestral sequences
  - Parameters, θ = {<mutation rates>, t_{ae}, t_{be}, t_{cf}, t_{df}, t_{eg}, t_{fg}}



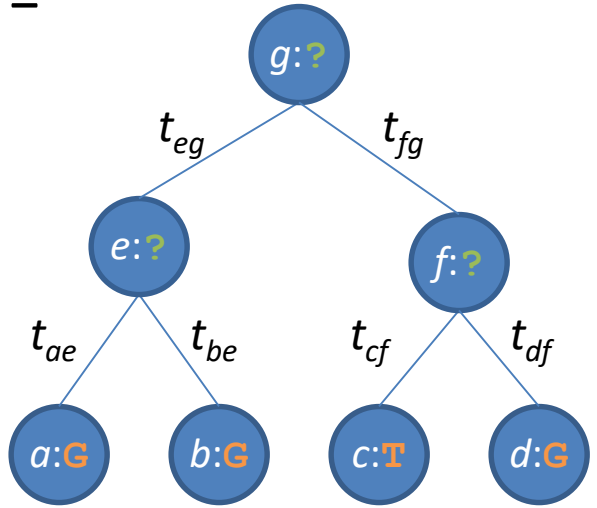
- Likelihood =Pr(g:G)
  Pr(e:G|g:G, t_{eg}) Pr(f:G|g:G, t_{fg})
  Pr(a:G|e:G, t_{ae}) Pr(b:G|e:G, t_{be})
  Pr(c:T|f:G, t_{cf}) Pr(d:G|f:G, t_{df})

  - We have learned how to compute these conditional probabilities for two mutation models (Jukes-Cantor and Kimura) in the last lecture

Computing likelihood

- In the small likelihood problem, we are only given the tree topology, but not the ancestral sequences – Then how to compute likelihood?
- Need to try them all (summation of $4^3 = 64$ terms): Likelihood =

$\Pr(g{:}\mathtt{A})$
$\Pr(e{:}\mathtt{A}|g{:}\mathtt{A}, t_{eg})$ $\Pr(f{:}\mathtt{A}|g{:}\mathtt{A}, t_{fg})$
$\Pr(a{:}\mathtt{G}|e{:}\mathtt{A}, t_{ae})$ $\Pr(b{:}\mathtt{G}|e{:}\mathtt{A}, t_{be})$
$\Pr(c{:}\mathtt{T}|f{:}\mathtt{A}, t_{cf})$ $\Pr(d{:}\mathtt{G}|f{:}\mathtt{A}, t_{df})$
+
$\Pr(g{:}\mathtt{C})$
$\Pr(e{:}\mathtt{A}|g{:}\mathtt{C}, t_{eg})$ $\Pr(f{:}\mathtt{A}|g{:}\mathtt{C}, t_{fg})$
$\Pr(a{:}\mathtt{G}|e{:}\mathtt{A}, t_{ae})$ $\Pr(b{:}\mathtt{G}|e{:}\mathtt{A}, t_{be})$
$\Pr(c{:}\mathtt{T}|f{:}\mathtt{A}, t_{cf})$ $\Pr(d{:}\mathtt{G}|f{:}\mathtt{A}, t_{df})$
+
…
+
$\Pr(g{:}\mathtt{T})$
$\Pr(e{:}\mathtt{T}|g{:}\mathtt{T}, t_{eg})$ $\Pr(f{:}\mathtt{T}|g{:}\mathtt{T}, t_{fg})$
$\Pr(a{:}\mathtt{G}|e{:}\mathtt{T}, t_{ae})$ $\Pr(b{:}\mathtt{G}|e{:}\mathtt{T}, t_{be})$
$\Pr(c{:}\mathtt{T}|f{:}\mathtt{T}, t_{cf})$ $\Pr(d{:}\mathtt{G}|f{:}\mathtt{T}, t_{df})$

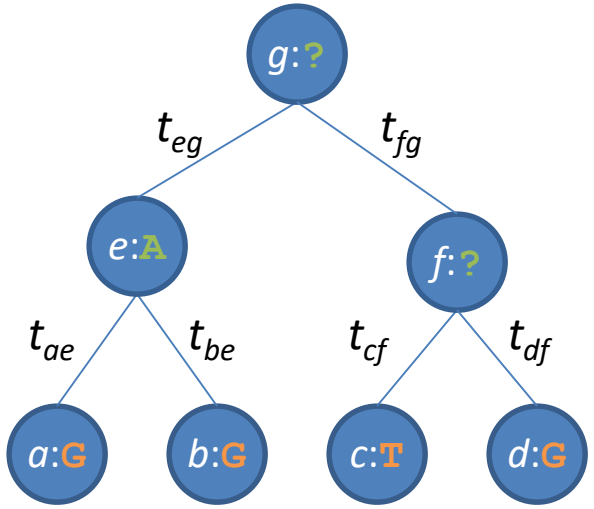


Possible ancestral states:

e	A	A	A	A	A		T
f	A	A	A	A	C	…	T
g	A	C	G	T	A		T

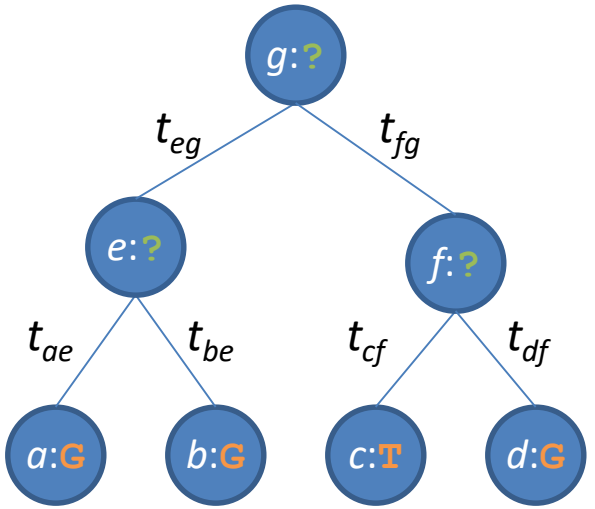
Computing likelihood efficiently [optional]

- An important observation: once the root of a sub-tree is determined, the likelihood of this sub-tree does not depend on other nodes in the whole tree
- E.g., once node *e* is decided to take character A, the likelihood of the sub-tree involving nodes *a*, *b* and *e* is
  $\Pr(e{:}\mathtt{A} \mid g, t_{eg})$
  $\Pr(a{:}\mathtt{G} \mid e{:}\mathtt{A}, t_{ae})\Pr(b{:}\mathtt{G} \mid e{:}\mathtt{A}, t_{be})$
  - If the character at node *g* does not change, the value of the above expression will not change no matter what character node *f* takes.
  - Therefore, this value can be re-used



Computing likelihood efficiently [optional]

- Define table V, where entry V(i,x) is the likelihood of the sub-tree rooted at i when the **parent** of i takes character x
  - Likelihood =
    Pr(g:A) V(e,A) V(f,A) +
    Pr(g:C) V(e,C) V(f,C) +
    Pr(g:G) V(e,G) V(f,G) +
    Pr(g:T) V(e,T) V(f,T)
  - V(e, A) =
    Pr(e:A | g:A,t_{eg}) V(a,A) V(b,A) +
    Pr(e:C | g:A,t_{eg}) V(a,C) V(b,C) +
    Pr(e:G | g:A,t_{eg}) V(a,G) V(b,G) +
    Pr(e:T | g:A,t_{eg}) V(a,T) V(b,T)
  - V(a,A) = Pr(a:G | e:A, t_{ae})
  - V(a,C) = Pr(a:G | e:C, t_{ae})
  - ...



- Table V contains O(n) entries. Computing the value for each entry requires a constant number of operations ⇒ Linear time overall

Solving the small likelihood problem

- Then how to find the optimal parameter values?
  - Start with a random estimate of θ
  - Apply a "hill climbing" algorithm
    - Change the value of a parameter so that the likelihood is increased
    - Repeat it for each parameter in turn, for multiple iterations
    - Will reach maximum if there is a single "peak" – This is true in many real situations, though theoretically cases can be constructed in which this it not true

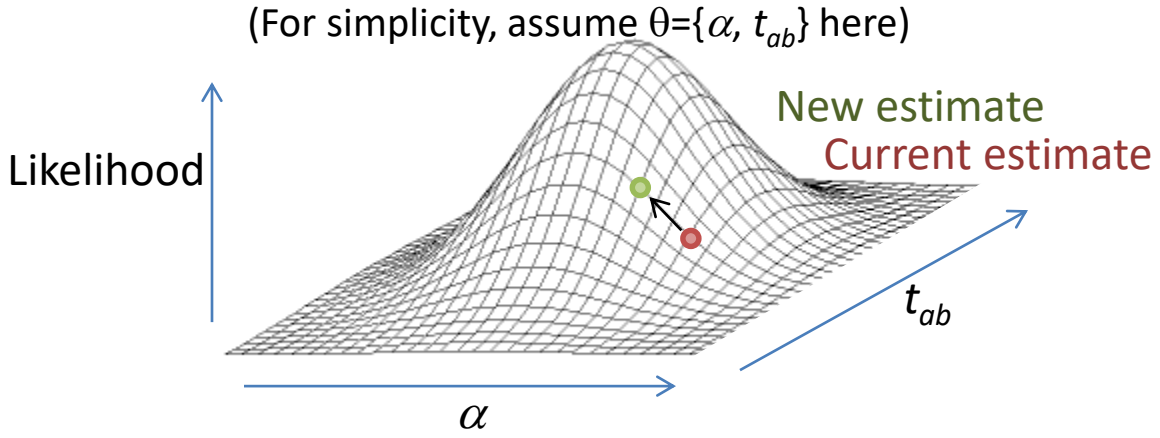


Image source: http://www.absoluteastronomy.com/topics/Hill_climbing

Which method to use?

- No definite answer
  - There are different camps
- In general, it is good to use methods that
  - Do not require strong assumptions
  - Are robust (do not produce drastically different results when the inputs are just slightly changed)
    - Build multiple trees using different parameters, then combine
    - Build trees with different subsets of sequences, then combine
    - Use probabilistic methods
  - Are computationally efficient
- There are many other algorithms that we did not cover, including those that consider mutation models.

Epilogue

Summary and Further Readings

Summary

- Two main types of tree reconstruction methods:

  - Distance-based
    - UPGMA
    - Neighbor joining

  - Sequence-based
    - Maximum parsimony
    - Maximum likelihood

Further readings

- Chapter 7 of *Algorithms in Bioinformatics: A Practical Introduction*
  - More details of the algorithms
  - Complexity analysis
  - Free slides available

Further readings

- Liu et al., *Science* 324(5934):1561-1564, (2009)
  - Although we have discussed multiple sequence alignment (MSA) and phylogenetic tree reconstruction in two different lectures, they are highly related
    - A good phylogenetic tree can guide the construction of the MSA
      - Recall the Clustal algorithm
    - A good MSA can help deduce the phylogenetic tree
      - For example, in computing sequence distances