CSI5126. Algorithms in bioinformatics

Phylogeny

Marcel Turcotte

uOttawa

School of Electrical Engineering and Computer Science (EECS)
University of Ottawa

Version October 11, 2018

Summary

In this module, we introduce **molecular evolution** concepts. Specifically, we consider building phylogentic trees. The general framework is two-step: **large phylogeny problem** and **small phylogeny problem**. We consider the three main approaches: **distance-based**, **character-based**, and **maximum likelihood**.

General objective

- **Explain in your own words** the three main approaches to building phylogenetic trees, with sufficient details so that an actual implementation can be made.

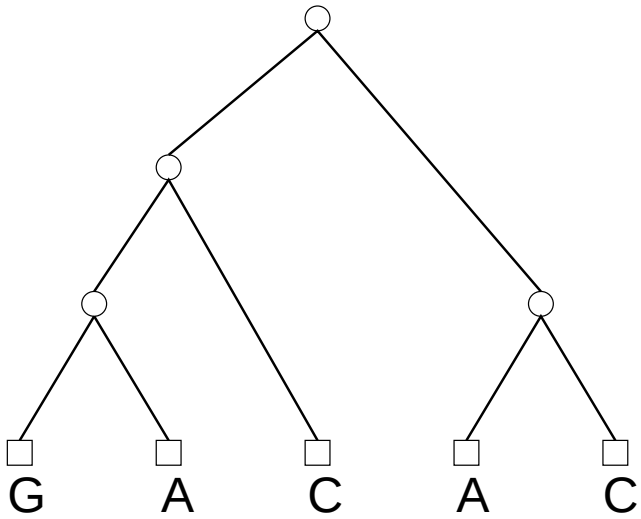
Reading

- Bernhard Haubold and Thomas Wiehe (2006). *Introduction to computational biology: an evolutionary approach*. Birkhäuser Basel. Pages 143-168.

II. **Character-based** tree reconstruction

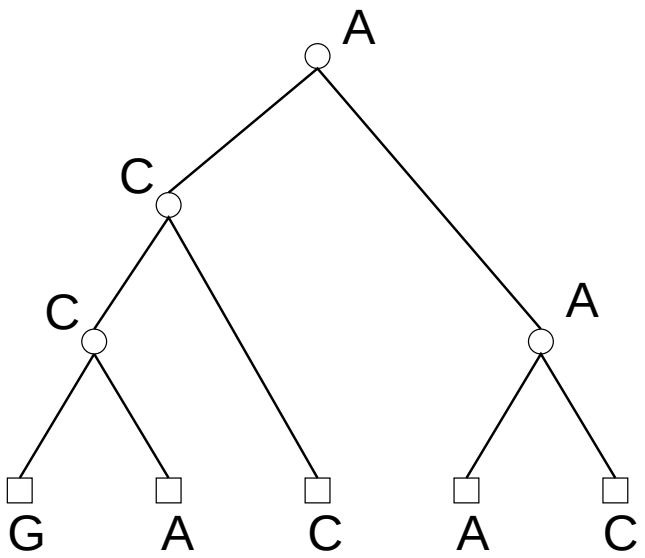
- Character-based reconstruction algorithms are labelling all the nodes of the tree with characters. **Leaves** are labelled with **observed data**. While the **internal nodes** are labelled with **hypothetical characters** (ancestral states).

Character-based tree reconstruction



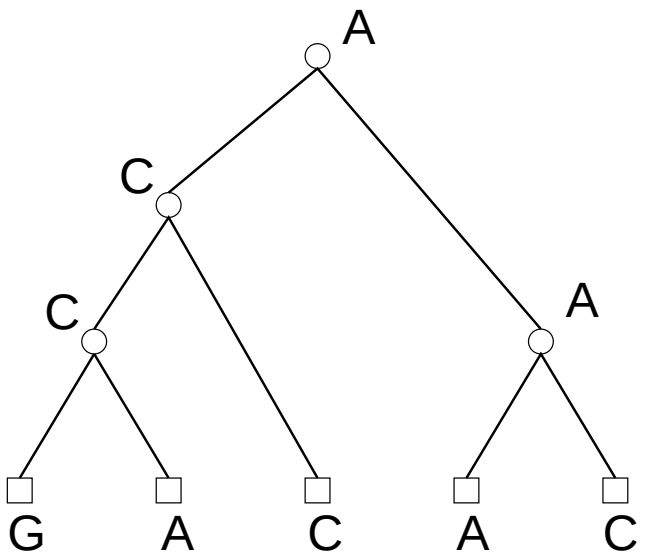
First, let's consider a **single character** (the *i*th nucleotide of a given gene in 5 species). The only observable characters are those at the leaves. Those correspond to the characters in todays organisms.

Character-based tree reconstruction



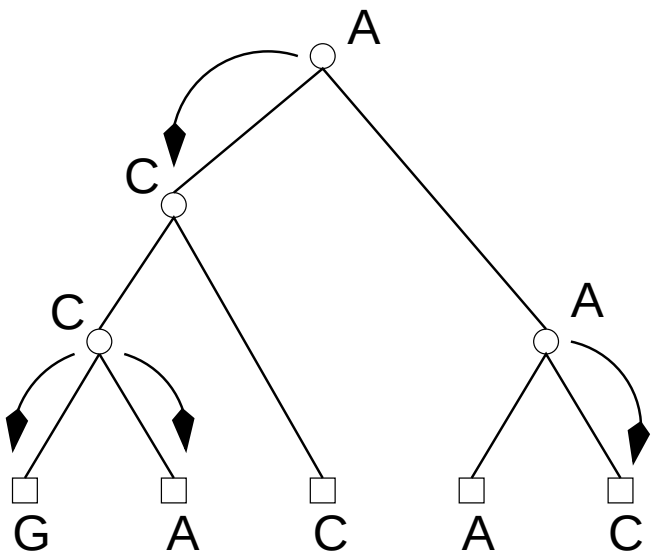
- Several reconstructions of the ancestral states are possible.

Character-based tree reconstruction



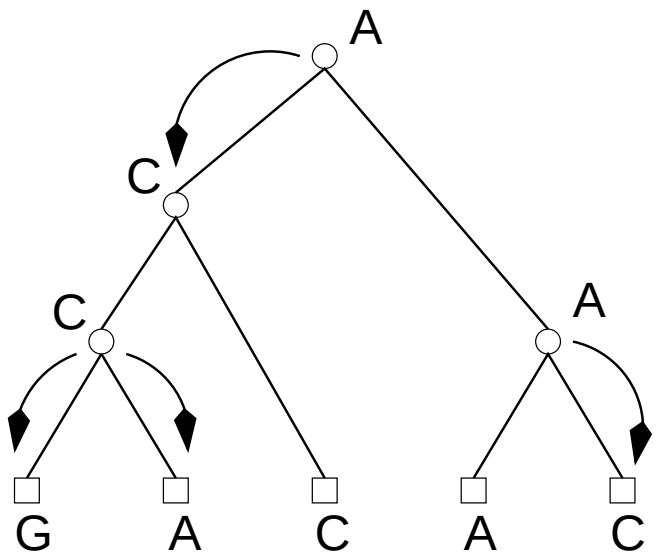
- Several reconstructions of the ancestral states are possible.
- **How many events** are represented on this tree?

Character-based tree reconstruction



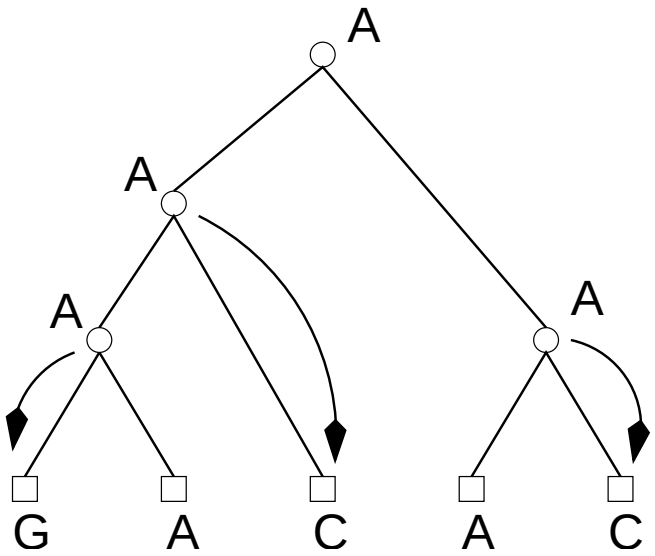
- The tree represents 4 events.

Character-based tree reconstruction



- The tree represents 4 events.
- Can you find a reconstruction that requires fewer events?

Character-based tree reconstruction



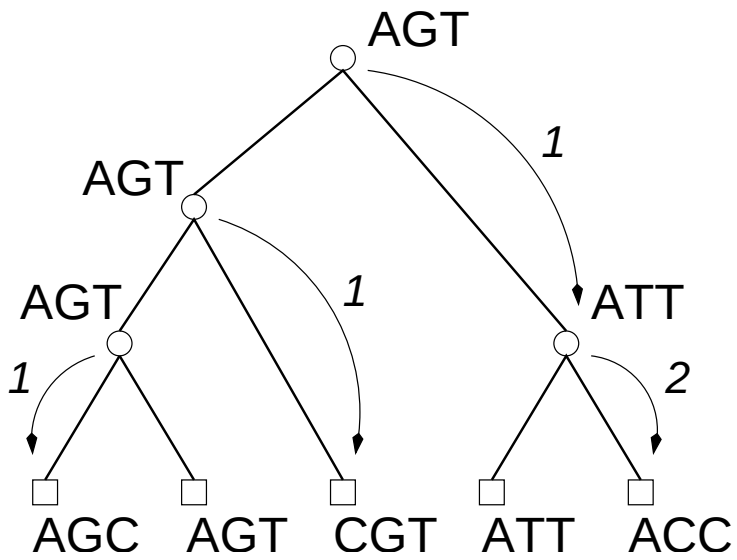
A 3 events tree.

Character-based tree reconstruction

Now considering **3 characters** (sites).

Species	**Sites** 1	2	3
A	A	G	C
B	A	G	T
C	C	G	T
D	A	T	T
E	A	C	C

Character-based tree reconstruction



A tree for five species and three characters. The reconstruction involves 5 mutations (evolutionary events).

Parsimony

- "Adoption of the **simplest assumption** in the formulation of a theory or in the interpretation of data, especially in accordance with the rule of **Ockham's razor**." **The American Heritage [Online] Dictionary**

Parsimony

- "Adoption of the **simplest assumption** in the formulation of a theory or in the interpretation of data, especially in accordance with the rule of **Ockham's razor**." **The American Heritage [Online] Dictionary**
- **Ockham's Razor**: "Plurality should not be posited without necessity."

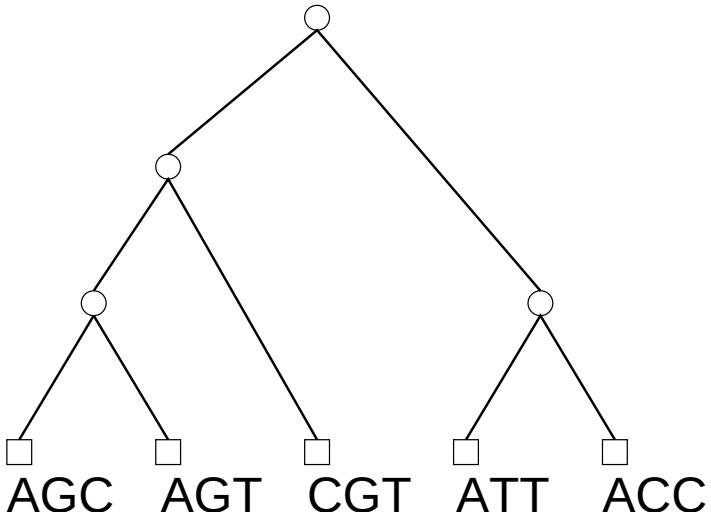
Parsimony

- "Adoption of the **simplest assumption** in the formulation of a theory or in the interpretation of data, especially in accordance with the rule of **Ockham's razor**." **The American Heritage [Online] Dictionary**
- **Ockham's Razor**: "Plurality should not be posited without necessity."
- "(1) **Mutations are exceedingly rare events** and (2) **the more unlikely events a model invokes, the less likely the model is to be correct.** As a result, the relationship that requires the fewest number of mutations to explain the current state of the sequences being considered is the relationship that is most likely to be correct." [3, page 98]

Issues

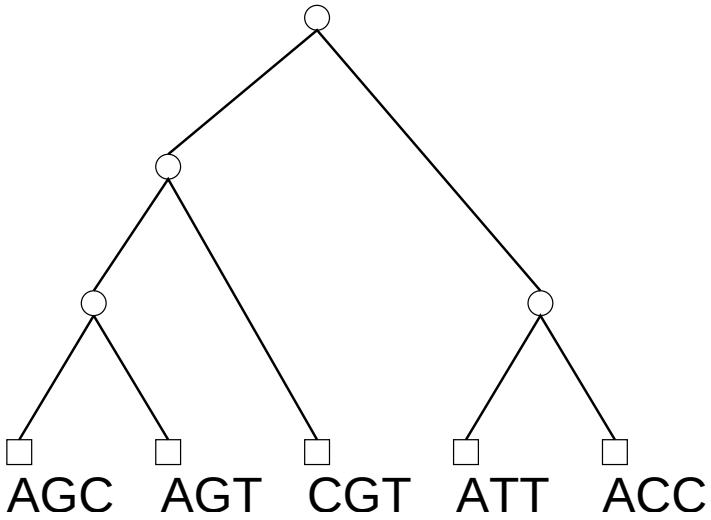
- Reconstructing the **ancestral states**;
- Counting the number of **changes**;
- **Find all** most parsimonious trees;
- Infer **branch lengths**;
- Is the most parsimonious tree the “real one”?
- Given several most parsimonious trees, is there a better one?

Small parsimony problem



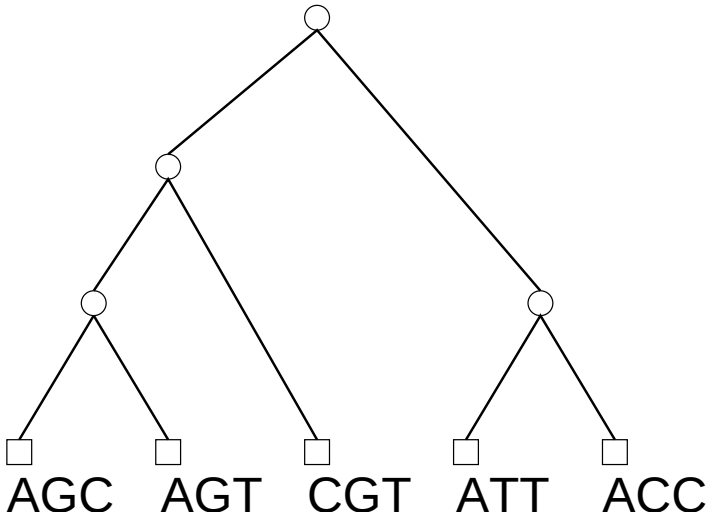
- **Problem:** *Find the most parsimonious labelling of the internal vertices in a given evolutionary tree.*

Small parsimony problem



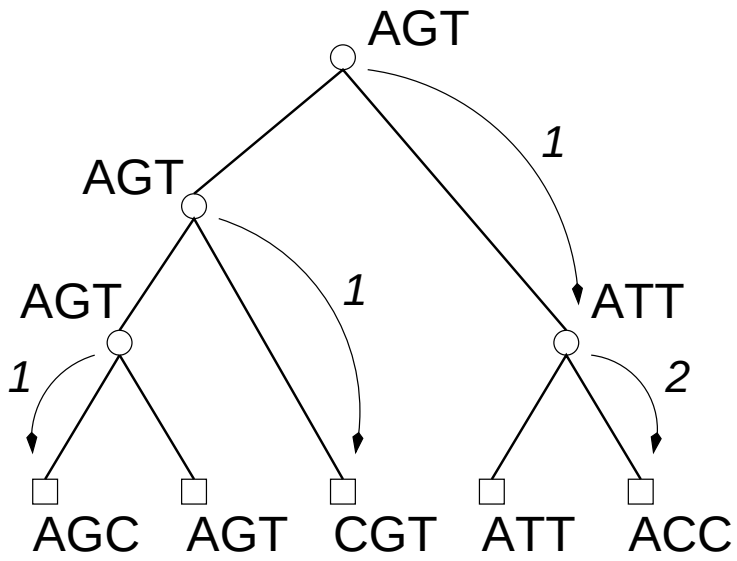
- **Problem:** *Find the most parsimonious labelling of the internal vertices in a given evolutionary tree.*
- **Input:** A tree T with each leaf labelled by an m-character array.

Small parsimony problem



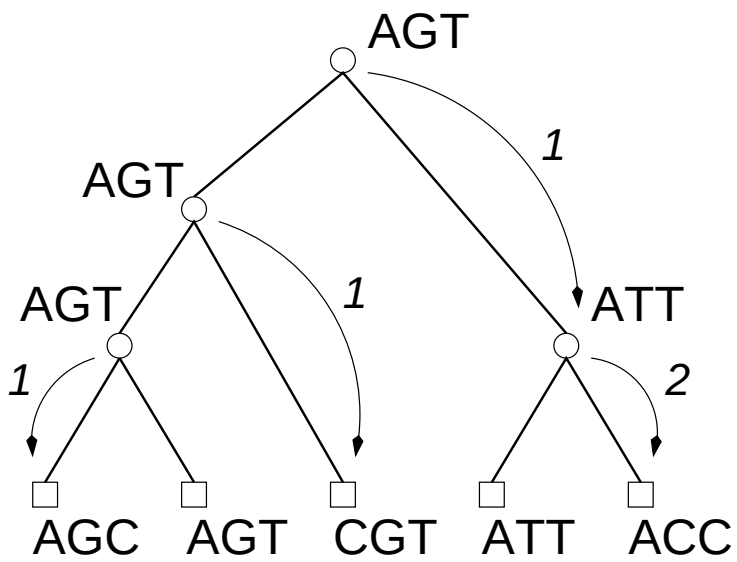
- **Problem:** *Find the most parsimonious labelling of the internal vertices in a given evolutionary tree.*
- **Input:** A tree T with each leaf labelled by an m-character array.
- **Output:** Labels (m-character arrays) for all the internal nodes such that $\Sigma d_H(u, v)$ for all the edges (u, v) is minimum; d_H is the Hamming distance.

Observation



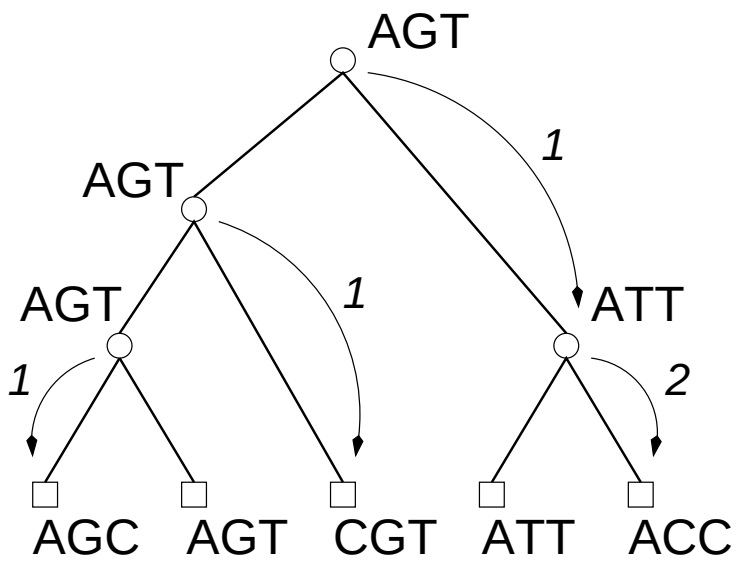
Notice that **the characters are independent**.

Observation



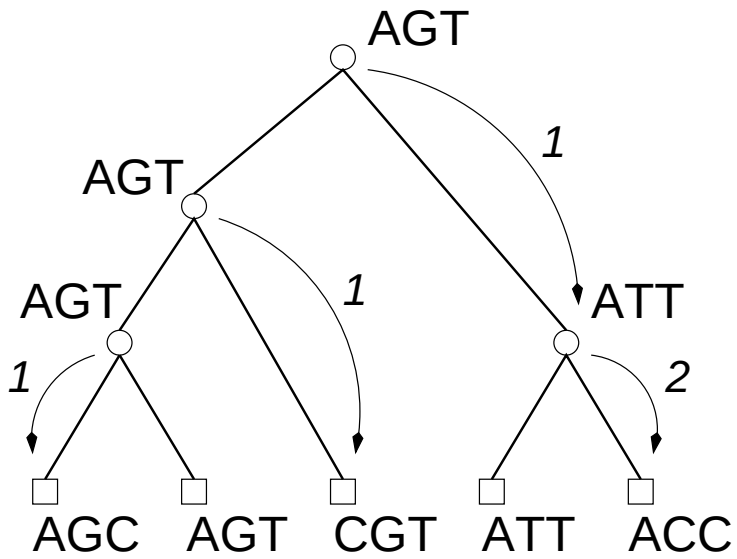
Notice that **the characters are independent**. The total number of changes is the **sum of** the number of changes for the first character, second character, and the third character.

Observation



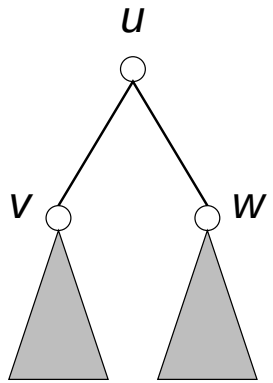
Notice that **the characters are independent**. The total number of changes is the **sum of** the number of changes for the first character, second character, and the third character. Thus, it suffices to develop a method that works for a **single character** and to apply it to all the characters.

Observation



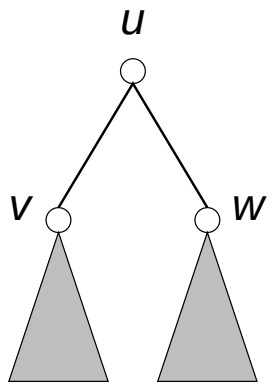
Notice that **the characters are independent**. The total number of changes is the **sum of** the number of changes for the first character, second character, and the third character. Thus, it suffices to develop a method that works for a **single character** and to apply it to all the characters. Proposals?

Small parsimony problem



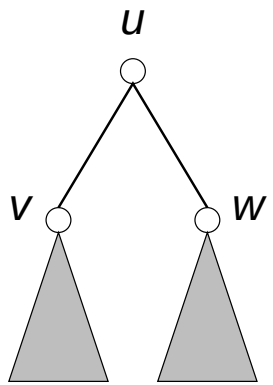
Let's define $s_c(u)$ as the **minimum parsimony score** obtained when u is labelled with c.

Small parsimony problem



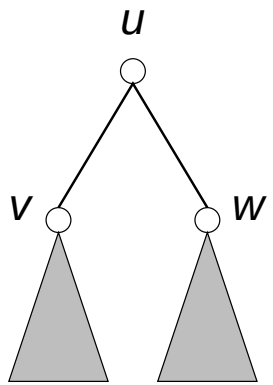
Let's define $s_c(u)$ as the **minimum parsimony score** obtained when u is labelled with c. How to computer $s_c(u)$?

Small parsimony problem



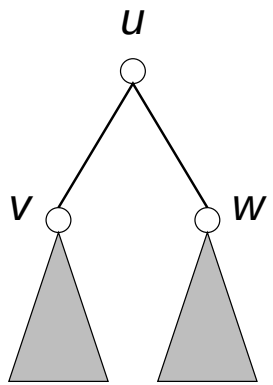
Let's define $s_c(u)$ as the **minimum parsimony score** obtained when u is labelled with c. How to computer $s_c(u)$? What do you need to know?

Small parsimony problem



Let's define $s_c(u)$ as the **minimum parsimony score** obtained when u is labelled with c. How to computer $s_c(u)$? What do you need to know? What are the dependencies?

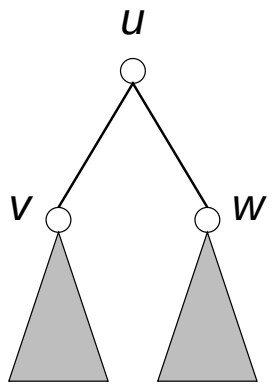
Small parsimony problem



Let's define $s_c(u)$ as the **minimum parsimony score** obtained when u is labelled with c. How to computer $s_c(u)$? What do you need to know? What are the dependencies?

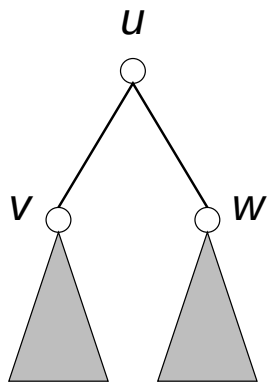
$$s_c(u) = \ldots$$

Small parsimony problem



For instance, what would be the most parsimonious score if u was labelled with A.

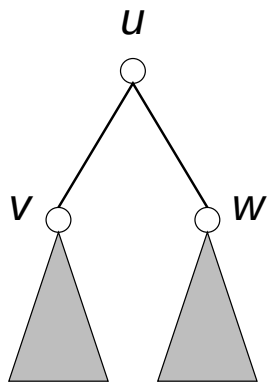
Small parsimony problem



For instance, what would be the most parsimonious score if u was labelled with A.

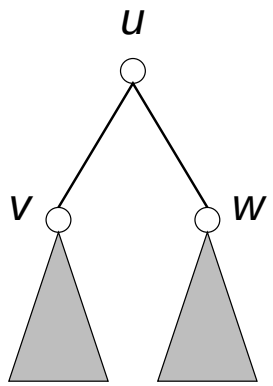
$$\begin{matrix} s_A(v)+? \\ s_C(v)+? \\ s_G(v)+? \\ s_T(v)+? \end{matrix} \quad ? \quad \begin{matrix} s_A(w)+? \\ s_C(w)+? \\ s_G(w)+? \\ s_T(w)+? \end{matrix}$$

Small parsimony problem



For instance, what would be the most parsimonious score if u was labelled with A.

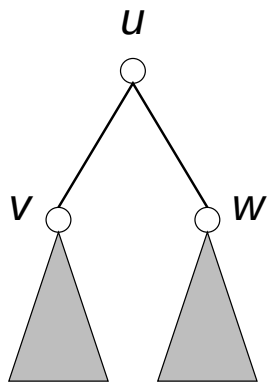
Small parsimony problem



For instance, what would be the most parsimonious score if u was labelled with A.

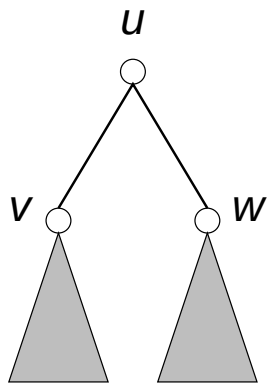
$$\begin{matrix} s_A(v)+0 \\ s_C(v)+1 \\ s_G(v)+1 \\ s_T(v)+1 \end{matrix} \quad ? \quad \begin{matrix} s_A(w)+0 \\ s_C(w)+1 \\ s_G(w)+1 \\ s_T(w)+1 \end{matrix}$$

Small parsimony problem



$$s_A(u) = \min \begin{cases} s_A(v) + 0 \\ s_C(v) + 1 \\ s_G(v) + 1 \\ s_T(v) + 1 \end{cases} + \min \begin{cases} s_A(w) + 0 \\ s_C(w) + 1 \\ s_G(w) + 1 \\ s_T(w) + 1 \end{cases}$$

Weighted small parsimony problem (Sankoff 1975)



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

where $\delta_{j,c}$ is a $k \times k$ scoring matrix.

Examples of **scoring matrices**

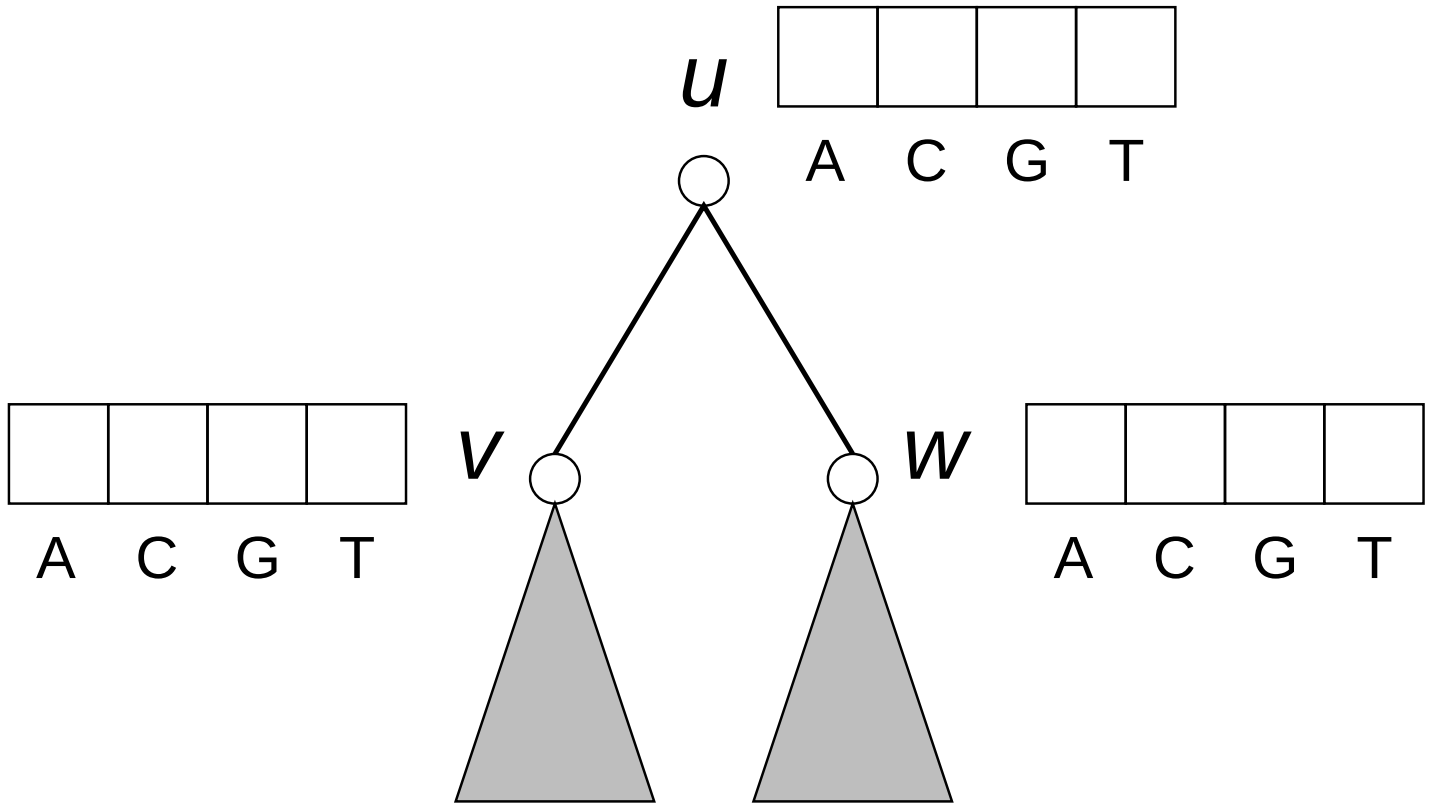
	A	**C**	**G**	**T**
A	0	1	1	1
C	1	0	1	1
G	1	1	0	1
T	1	1	1	0

	A	**C**	**G**	**T**
A	0	1	0.33	1
C	1	0	1	0.33
G	0.33	1	0	1
T	1	0.33	1	0

Solving the small parsimony problem

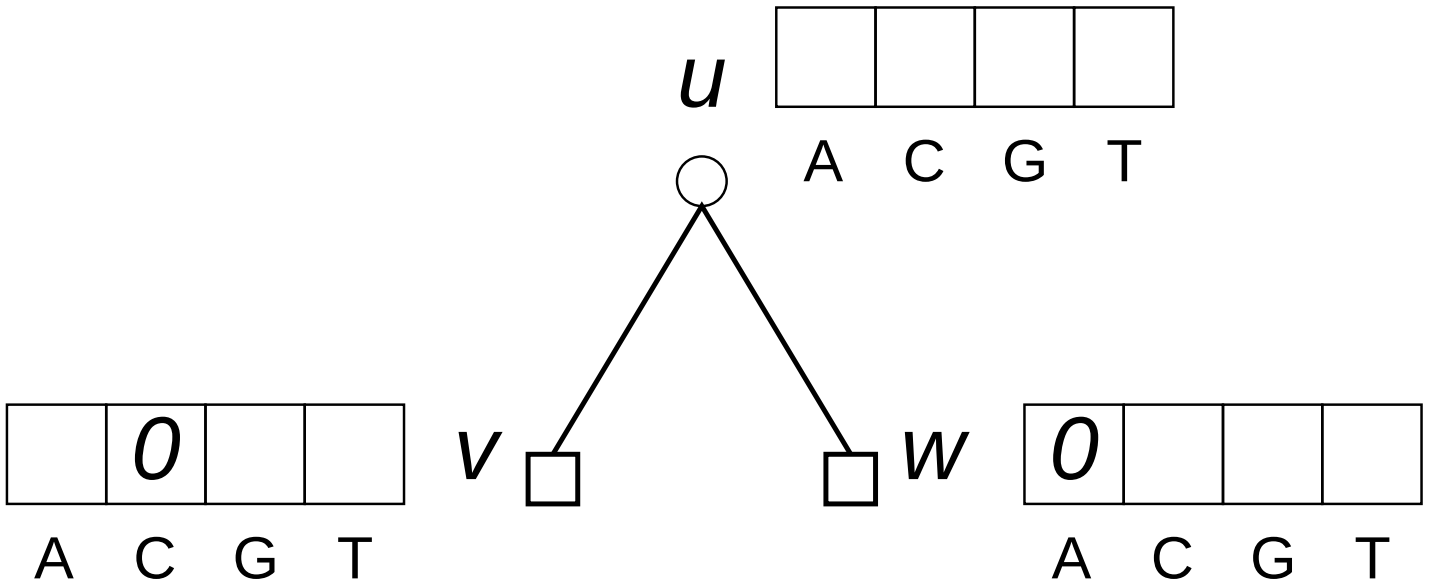
General case.

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

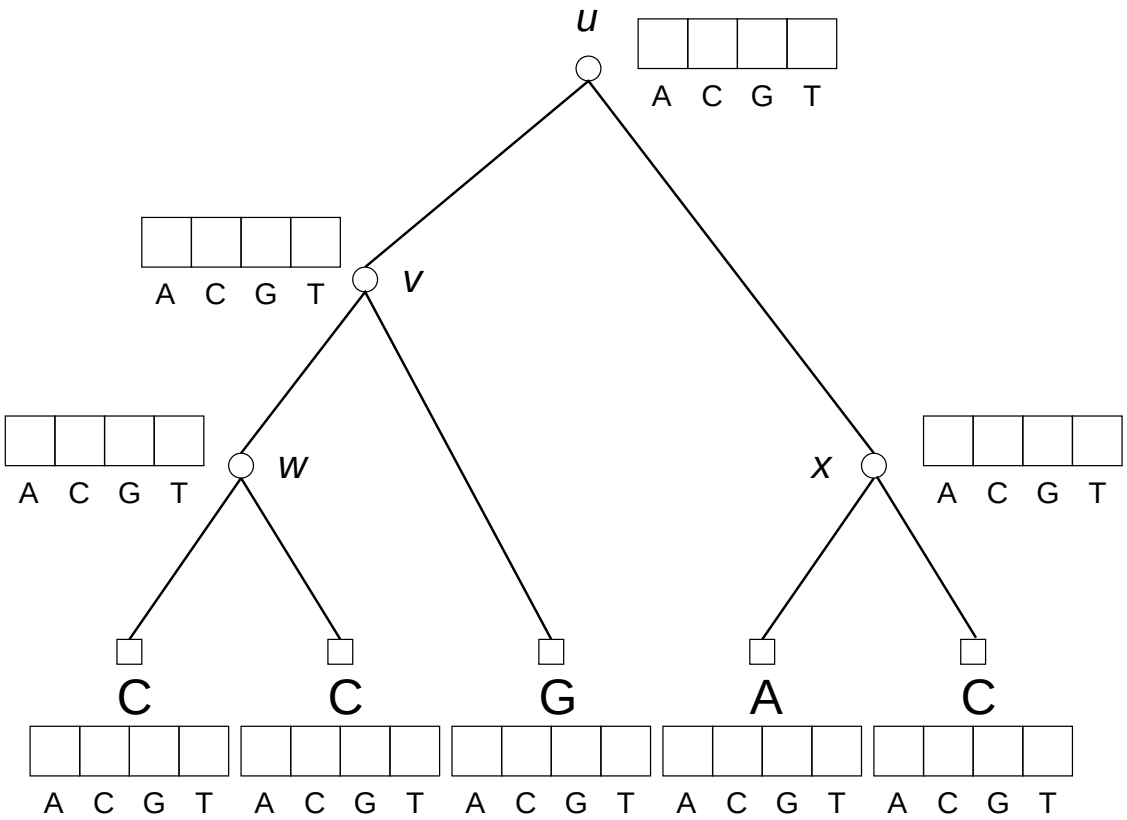


Solving the small parsimony problem

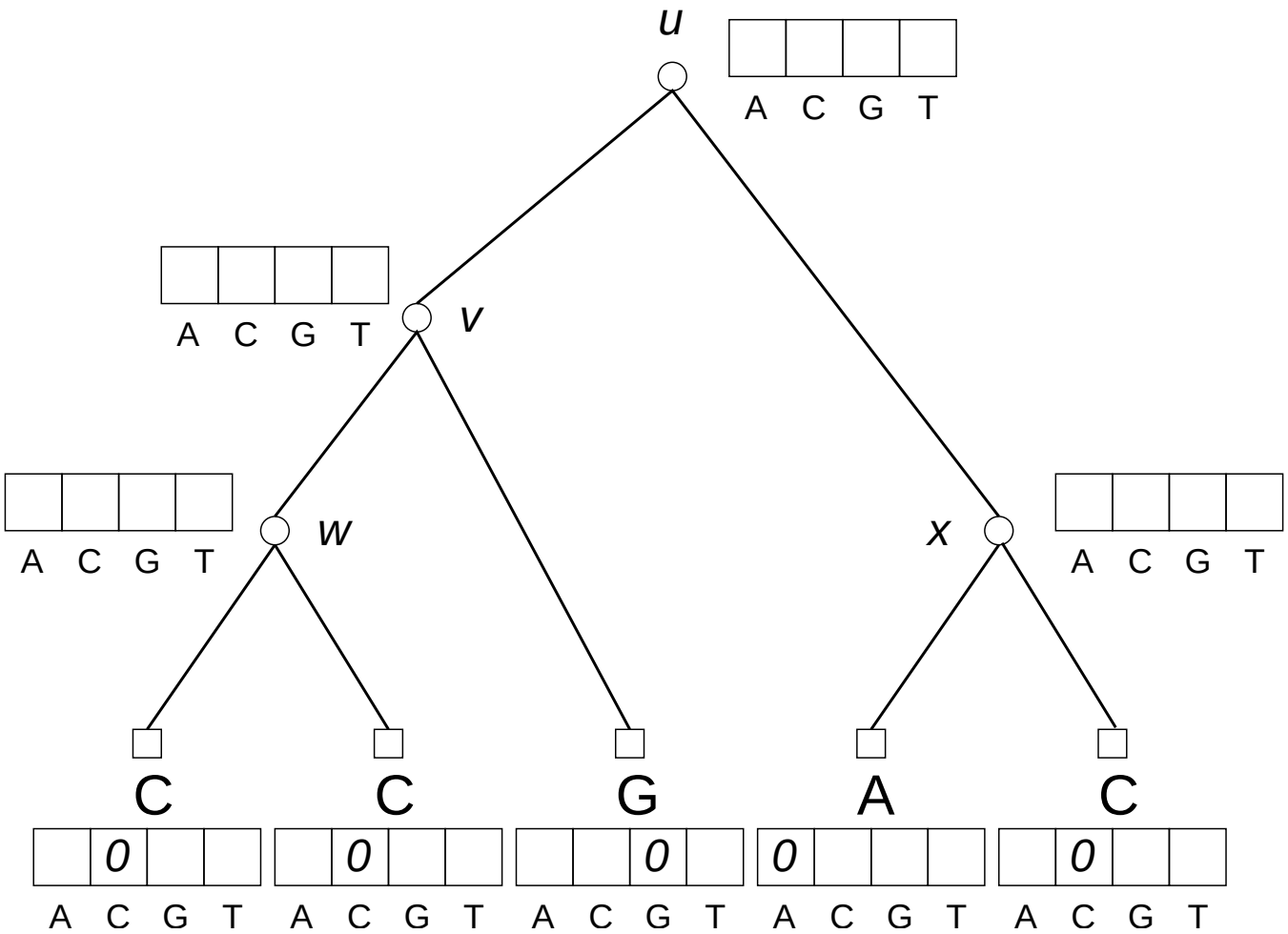
Initialisation.
For each leaf, $s_c(v) = 0$ if character c is found at that node and ∞ otherwise.



Small parsimony problem



Small parsimony problem



$$S_A(w) = 2 = \min\{\infty + 0, 0 + 1, \infty + 1, \infty + 1\} + \min\{\infty + 0, 0 + 1, \infty + 1, \infty + 1\}$$
$$S_C(w) = 0 = \min\{\infty + 1, 0 + 0, \infty + 1, \infty + 1\} + \min\{\infty + 1, 0 + 0, \infty + 1, \infty + 1\}$$
$$S_G(w) = 2 = \min\{\infty + 1, 0 + 1, \infty + 0, \infty + 1\} + \min\{\infty + 1, 0 + 1, \infty + 0, \infty + 1\}$$
$$S_T(w) = 2 = \min\{\infty + 1, 0 + 1, \infty + 1, \infty + 0\} + \min\{\infty + 1, 0 + 1, \infty + 1, \infty + 0\}$$

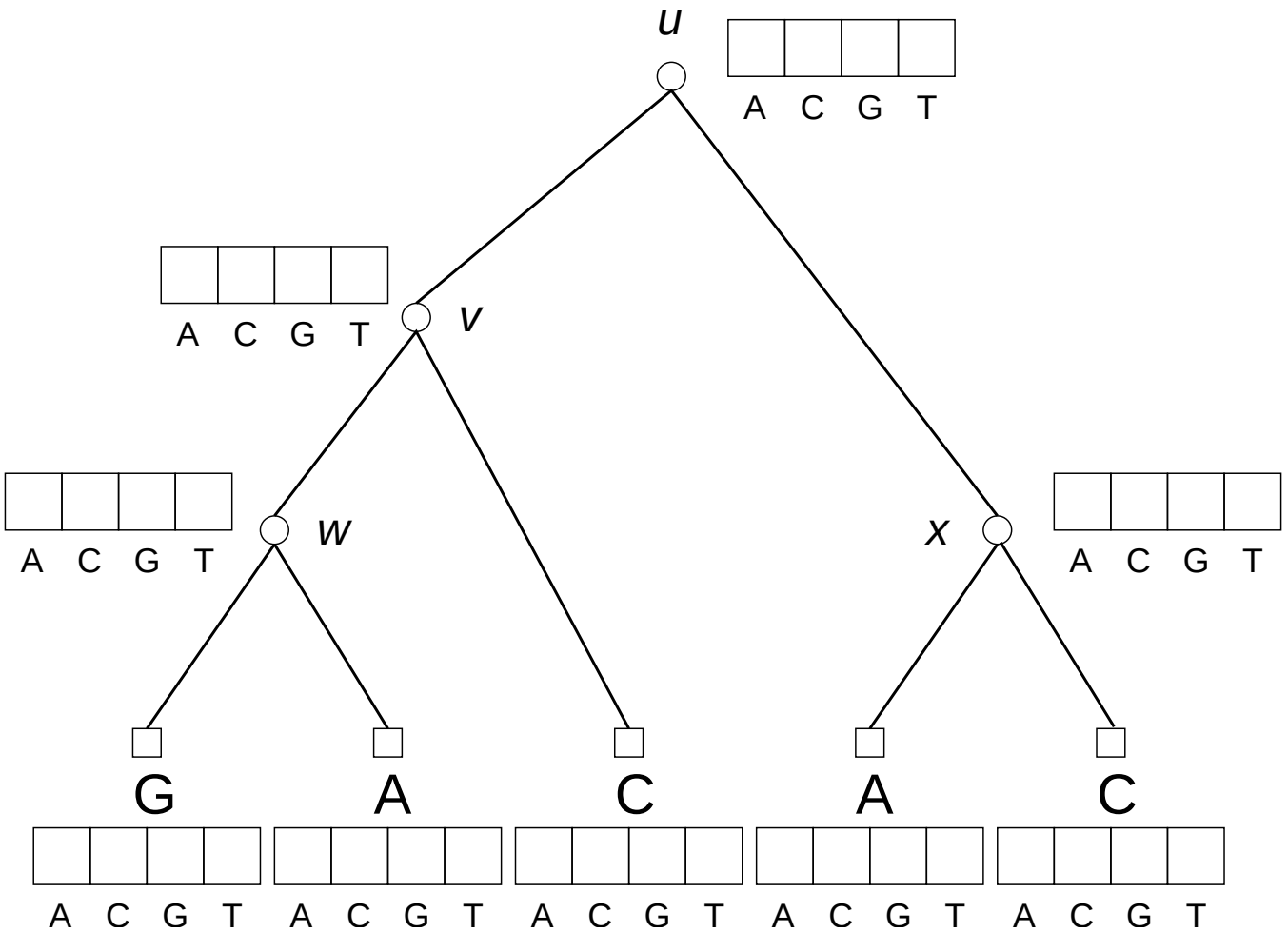
$$S_A(x) = 1 = \min\{0 + 0, \infty + 1, \infty + 1, \infty + 1\} + \min\{\infty + 0, 0 + 1, \infty + 1, \infty + 1\}$$
$$S_C(x) = 1 = \min\{0 + 1, \infty + 0, \infty + 1, \infty + 1\} + \min\{\infty + 1, 0 + 0, \infty + 1, \infty + 1\}$$
$$S_G(x) = 2 = \min\{0 + 1, \infty + 1, \infty + 0, \infty + 1\} + \min\{\infty + 1, 0 + 1, \infty + 0, \infty + 1\}$$
$$S_T(x) = 2 = \min\{0 + 1, \infty + 1, \infty + 1, \infty + 0\} + \min\{\infty + 1, 0 + 1, \infty + 1, \infty + 0\}$$

$$S_A(v) = 2 = \min\{2 + 0, 0 + 1, 2 + 1, 2 + 1\} + \min\{\infty + 0, \infty + 1, 0 + 1, \infty + 1\}$$
$$S_C(v) = 1 = \min\{2 + 1, 0 + 0, 2 + 1, 2 + 1\} + \min\{\infty + 1, \infty + 0, 0 + 1, \infty + 1\}$$
$$S_G(v) = 1 = \min\{2 + 1, 0 + 1, 2 + 0, 2 + 1\} + \min\{\infty + 1, \infty + 1, 0 + 0, \infty + 1\}$$
$$S_T(v) = 2 = \min\{2 + 1, 0 + 1, 2 + 1, 2 + 0\} + \min\{\infty + 1, \infty + 1, 0 + 1, \infty + 0\}$$

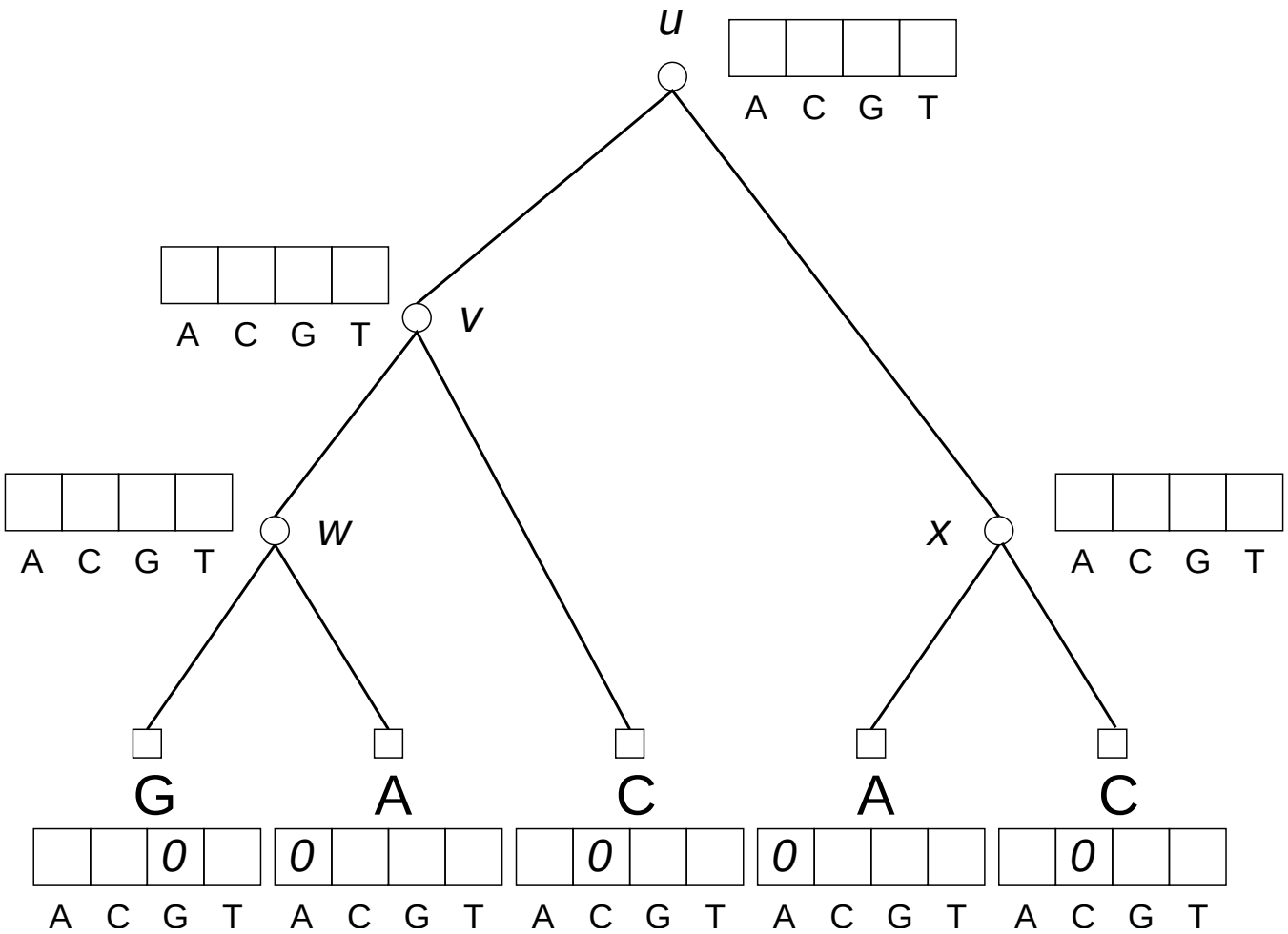
$$S_A(u) = 3 = \min\{2 + 0, 1 + 1, 1 + 1, 2 + 1\} + \min\{1 + 0, 1 + 1, 2 + 1, 2 + 1\}$$
$$S_C(u) = 2 = \min\{2 + 1, 1 + 0, 1 + 1, 2 + 1\} + \min\{1 + 1, 1 + 0, 2 + 1, 2 + 1\}$$
$$S_G(u) = 3 = \min\{2 + 1, 1 + 1, 1 + 0, 2 + 1\} + \min\{1 + 1, 1 + 1, 2 + 0, 2 + 1\}$$
$$S_T(u) = 4 = \min\{2 + 1, 1 + 1, 1 + 1, 2 + 0\} + \min\{1 + 1, 1 + 1, 2 + 1, 2 + 0\}$$

- Is the solution **unique**?
- How do you **retrieve** a solution?

Small parsimony problem



Small parsimony problem



$$S_A(w) = 1 = \min\{\infty + 0, \infty + 1, 0 + 1, \infty + 1\} + \min\{0 + 0, \infty + 1, \infty + 1, \infty + 1\}$$
$$S_C(w) = 2 = \min\{\infty + 1, \infty + 0, 0 + 1, \infty + 1\} + \min\{0 + 1, \infty + 0, \infty + 1, \infty + 1\}$$
$$S_G(w) = 1 = \min\{\infty + 1, \infty + 1, 0 + 0, \infty + 1\} + \min\{0 + 1, \infty + 1, \infty + 0, \infty + 1\}$$
$$S_T(w) = 2 = \min\{\infty + 1, \infty + 1, 0 + 1, \infty + 0\} + \min\{0 + 1, \infty + 1, \infty + 1, \infty + 0\}$$

$$S_A(x) = 1 = \min\{0 + 0, \infty + 1, \infty + 1, \infty + 1\} + \min\{\infty + 0, 0 + 1, \infty + 1, \infty + 1\}$$
$$S_C(x) = 1 = \min\{0 + 1, \infty + 0, \infty + 1, \infty + 1\} + \min\{\infty + 1, 0 + 0, \infty + 1, \infty + 1\}$$
$$S_G(x) = 2 = \min\{0 + 1, \infty + 1, \infty + 0, \infty + 1\} + \min\{\infty + 1, 0 + 1, \infty + 0, \infty + 1\}$$
$$S_T(x) = 2 = \min\{0 + 1, \infty + 1, \infty + 1, \infty + 0\} + \min\{\infty + 1, 0 + 1, \infty + 1, \infty + 0\}$$

$$S_A(v) = 2 = \min\{1 + 0, 2 + 1, 1 + 1, 2 + 1\} + \min\{\infty + 0, 0 + 1, \infty + 1, \infty + 1\}$$
$$S_C(v) = 2 = \min\{1 + 1, 2 + 0, 1 + 1, 2 + 1\} + \min\{\infty + 1, 0 + 0, \infty + 1, \infty + 1\}$$
$$S_G(v) = 2 = \min\{1 + 1, 2 + 1, 1 + 0, 2 + 1\} + \min\{\infty + 1, 0 + 1, \infty + 0, \infty + 1\}$$
$$S_T(v) = 3 = \min\{1 + 1, 2 + 1, 1 + 1, 2 + 0\} + \min\{\infty + 1, 0 + 1, \infty + 1, \infty + 0\}$$

$$S_A(u) = 3 = \min\{2 + 0, 2 + 1, 2 + 1, 3 + 1\} + \min\{1 + 0, 1 + 1, 2 + 1, 2 + 1\}$$
$$S_C(u) = 3 = \min\{2 + 1, 2 + 0, 2 + 1, 3 + 1\} + \min\{1 + 1, 1 + 0, 2 + 1, 2 + 1\}$$
$$S_G(u) = 4 = \min\{2 + 1, 2 + 1, 2 + 0, 3 + 1\} + \min\{1 + 1, 1 + 1, 2 + 0, 2 + 1\}$$
$$S_T(u) = 5 = \min\{2 + 1, 2 + 1, 2 + 1, 3 + 0\} + \min\{1 + 1, 1 + 1, 2 + 1, 2 + 0\}$$

Large parsimony problem

- **Problem:** *Find a tree having the minimum parsimony score.*

Large parsimony problem

- **Problem:** *Find a tree having the minimum parsimony score.*
- **Input:** An $n \times m$ matrix (alignment).

Large parsimony problem

- **Problem:** *Find a tree having the minimum parsimony score.*
- **Input:** An $n \times m$ matrix (alignment).
- **Output:** A tree T with n leaves labeled by the n rows (m characters) of the input matrix. The internal nodes are labelled with arrays of m characters such that the overall parsimony score is minimum.

Large parsimony problem

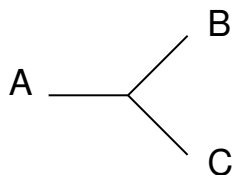
- **Problem:** *Find a tree having the minimum parsimony score.*
- **Input:** An $n \times m$ matrix (alignment).
- **Output:** A tree T with n leaves labeled by the n rows (m characters) of the input matrix. The internal nodes are labelled with arrays of m characters such that the overall parsimony score is minimum.
- The problem is known to be $\mathcal{NP}$-complete.

Exhaustive approach: 4 to 15 sequences

# Species	# unrooted trees
4	3
5	15
6	105
7	945
8	10,395
9	1,35,135
10	2,027,025
11	34,459,425
12	654,729,075
13	13,749,310,575
14	316,234,143,225
15	7,905,853,580,625

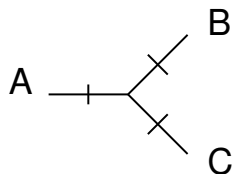
For a small number of species, say less than 15, it is be possible to exhaustively enumerate all the trees, and for each tree calculate the minimum parsimony score. The tree that has the overall minimum parsimony score is reported.

Sequential addition strategy



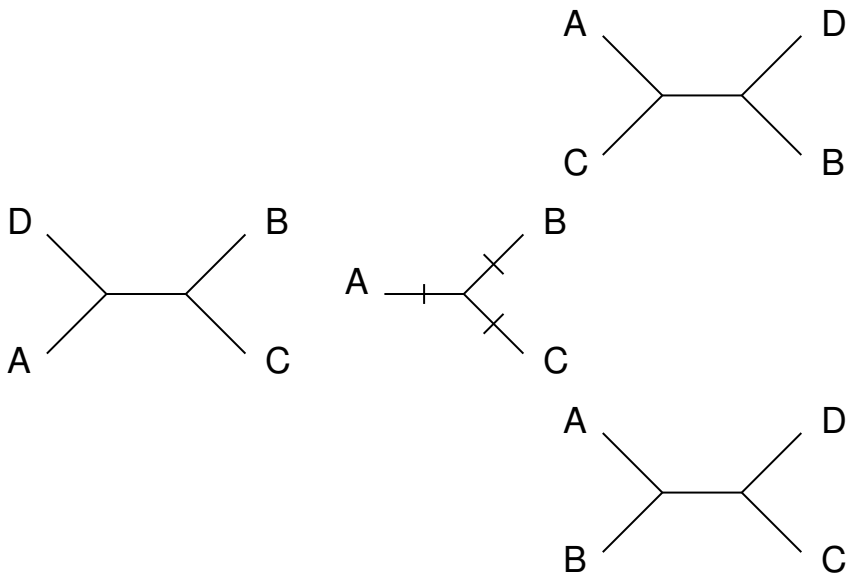
Given three species, there is a single unrooted tree.

Sequential addition strategy



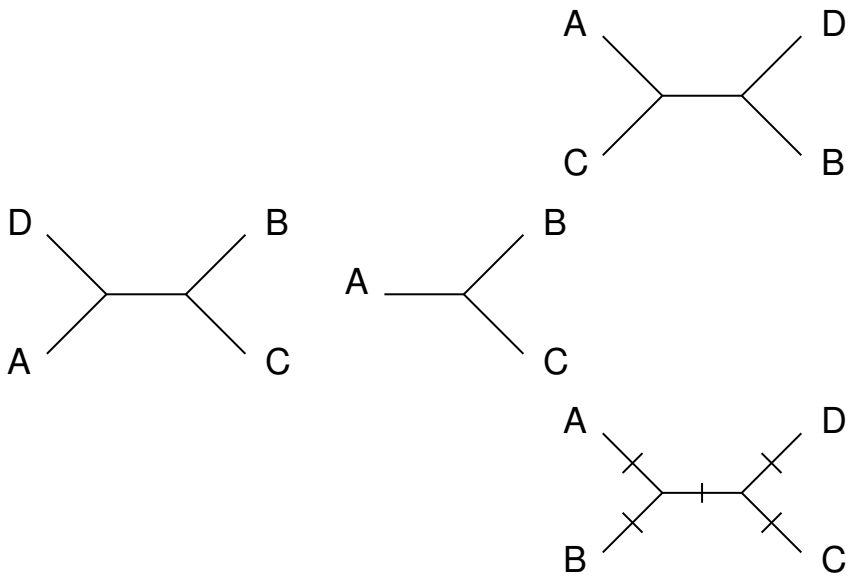
Each branch can serve as an insertion point, adding a new branch off the middle of any existing branch.

Sequential addition strategy



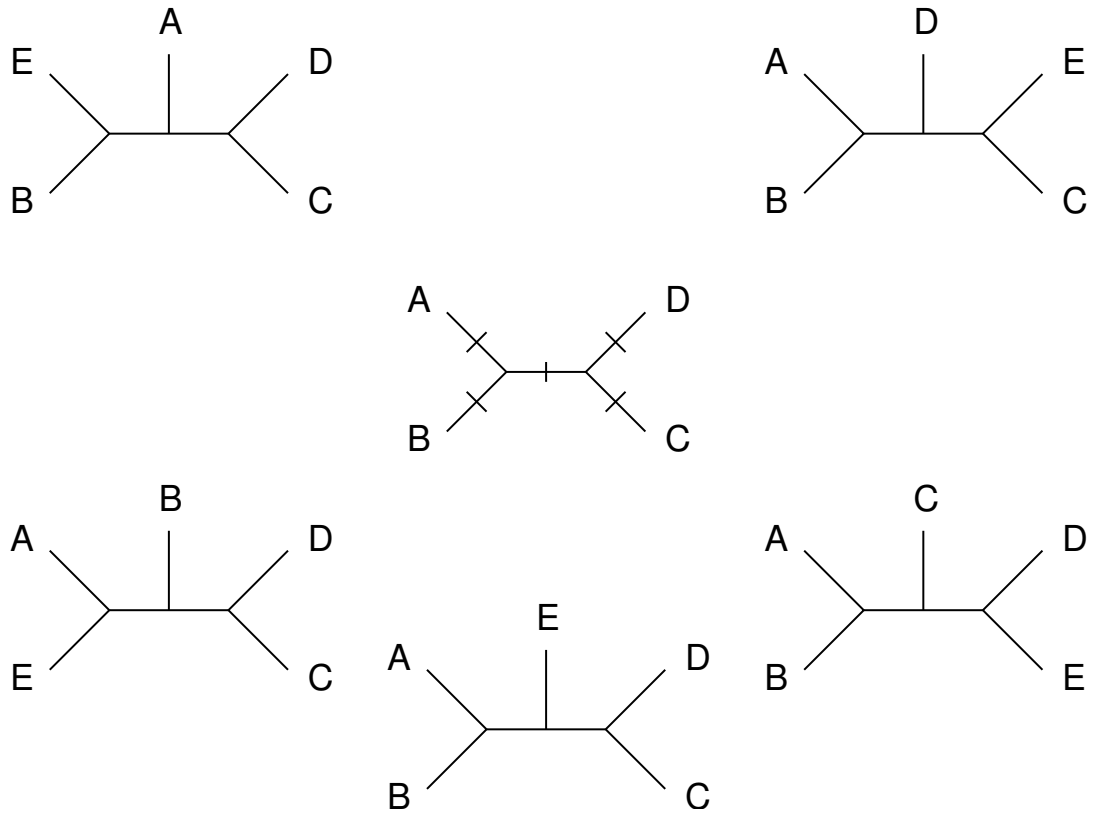
Therefore producing 3 four species unrooted trees.

Sequential addition strategy



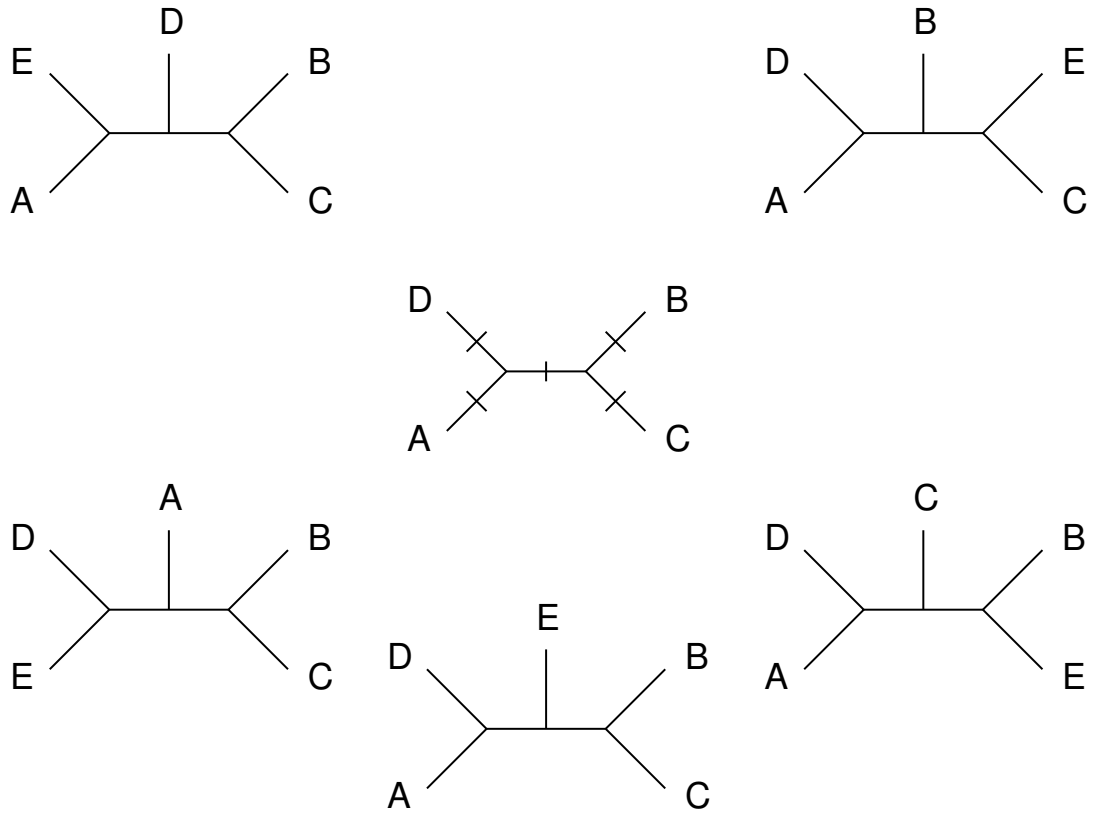
The same process is applied to all 3 four species trees.

Sequential addition strategy



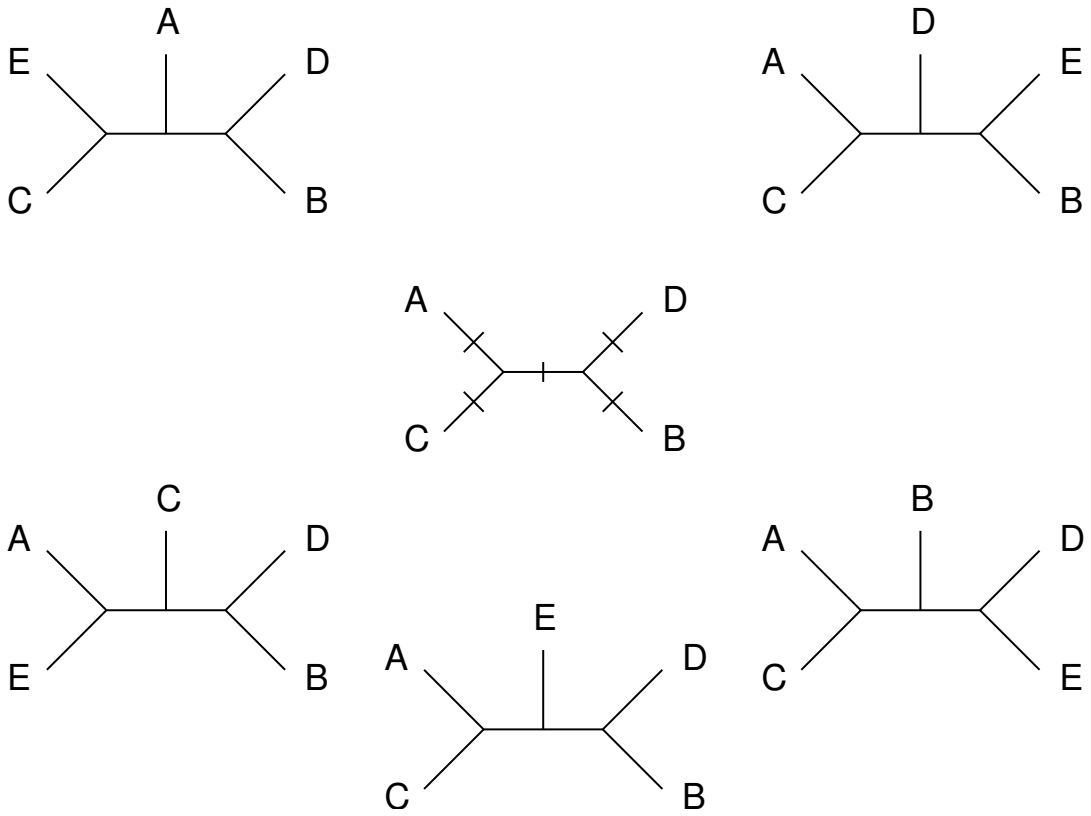
A four species unrooted tree has 5 edges, thus leading to 5 new unrooted trees.

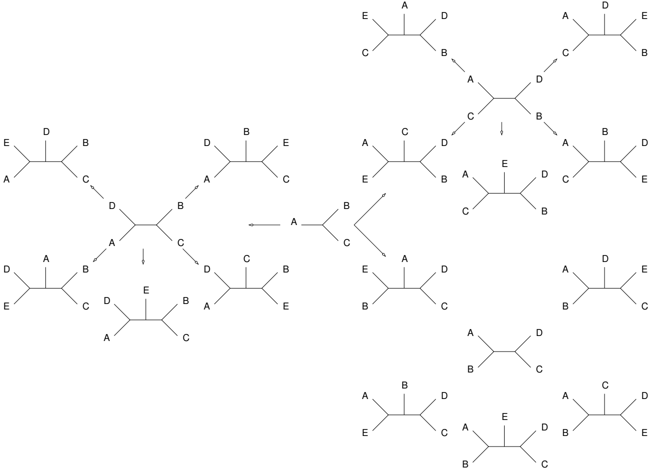
Sequential addition strategy



There will be 15 five species unrooted trees.

Sequential addition strategy





Branch and bound

- Just like **backtracking**, **branch and bound** is a **state space search** algorithm.

Branch and bound

- Just like **backtracking**, **branch and bound** is a **state space search** algorithm.
- **Branch and bound** is used to solve **optimization** problems. Herein, for simplicity, let's assume a **minimization** problem is to be solved.

Branch and bound

- Just like **backtracking**, **branch and bound** is a **state space search** algorithm.
- **Branch and bound** is used to solve **optimization** problems. Herein, for simplicity, let's assume a **minimization** problem is to be solved.
- Just like **backtracking**, **non-promising** nodes, and their descendants, are **pruned** from the search space.

Branch and bound

- Just like **backtracking**, **branch and bound** is a **state space search** algorithm.
- **Branch and bound** is used to solve **optimization** problems. Herein, for simplicity, let's assume a **minimization** problem is to be solved.
- Just like **backtracking**, **non-promising** nodes, and their descendants, are **pruned** from the search space.
- **For each node**, the algorithm computes a **bound**.

Branch and bound

- Just like **backtracking**, **branch and bound** is a **state space search** algorithm.
- **Branch and bound** is used to solve **optimization** problems. Herein, for simplicity, let's assume a **minimization** problem is to be solved.
- Just like **backtracking**, **non-promising** nodes, and their descendants, are **pruned** from the search space.
- **For each node**, the algorithm computes a **bound**.
  - The bound generally consists of **two terms**: the cost for the **partial solution** up to that node, as well as, a lower bound for the minimum cost **extending the solution** (visiting the yet unseen states).

Branch and bound

- Just like **backtracking**, **branch and bound** is a **state space search** algorithm.
- **Branch and bound** is used to solve **optimization** problems. Herein, for simplicity, let's assume a **minimization** problem is to be solved.
- Just like **backtracking**, **non-promising** nodes, and their descendants, are **pruned** from the search space.
- **For each node**, the algorithm computes a **bound**.
  - The bound generally consists of **two terms**: the cost for the **partial solution** up to that node, as well as, a lower bound for the minimum cost **extending the solution** (visiting the yet unseen states).
  - The descendants of a node are **pruned** (not visited), if the **best solution** that could be found in the sub-tree would be **worse** than the **best** solution found so far.

Branch and bound (continued)

- **No** prescribed order to search the tree:

Branch and bound (continued)

- **No** prescribed order to search the tree:
  - **Queue: breadth-first** search with branch and bound pruning;

Branch and bound (continued)

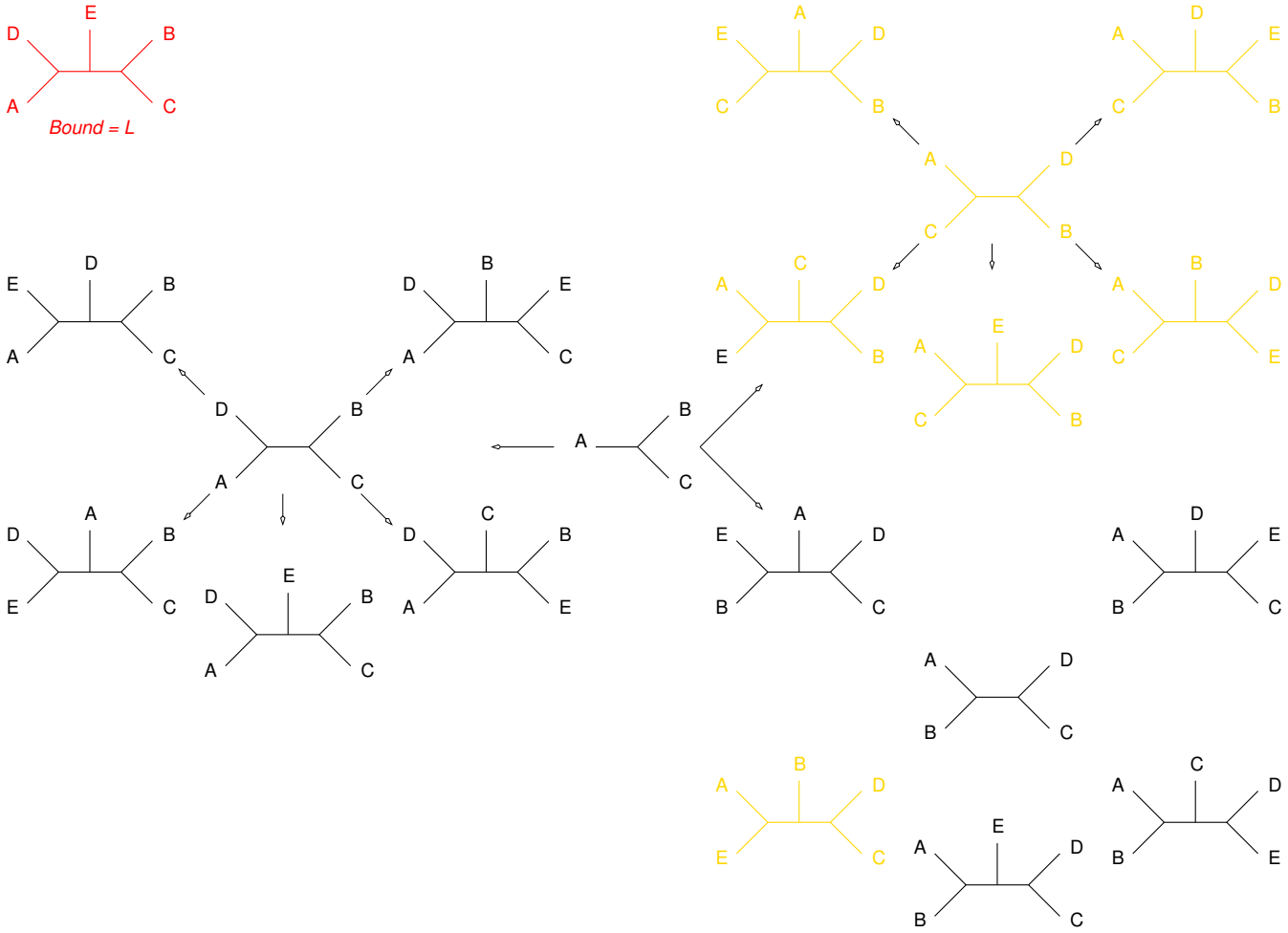
- **No** prescribed order to search the tree:
  - **Queue: breadth-first** search with branch and bound pruning;
  - **Stack: depth-first** search with branch and bound pruning;

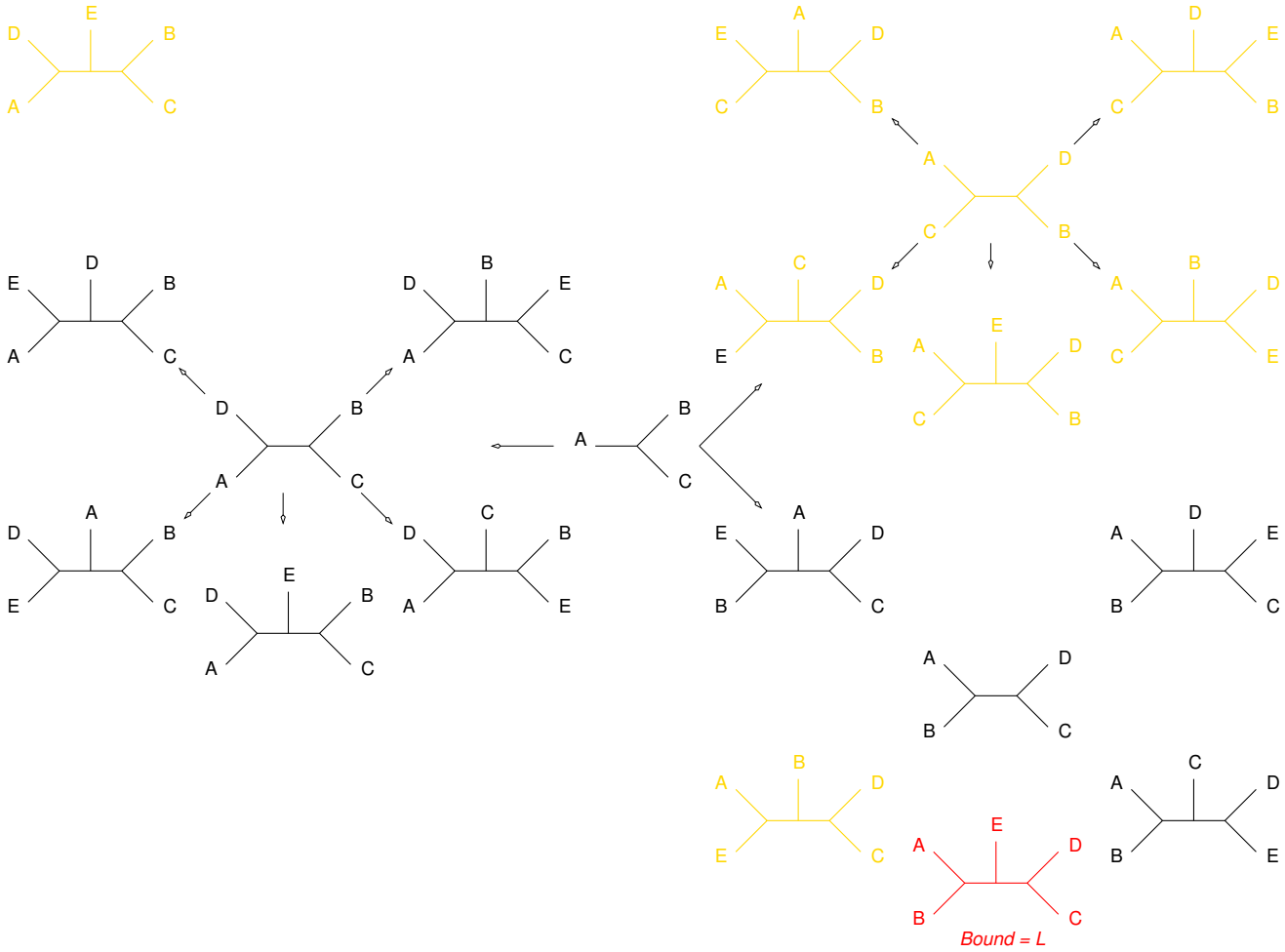
Branch and bound (continued)

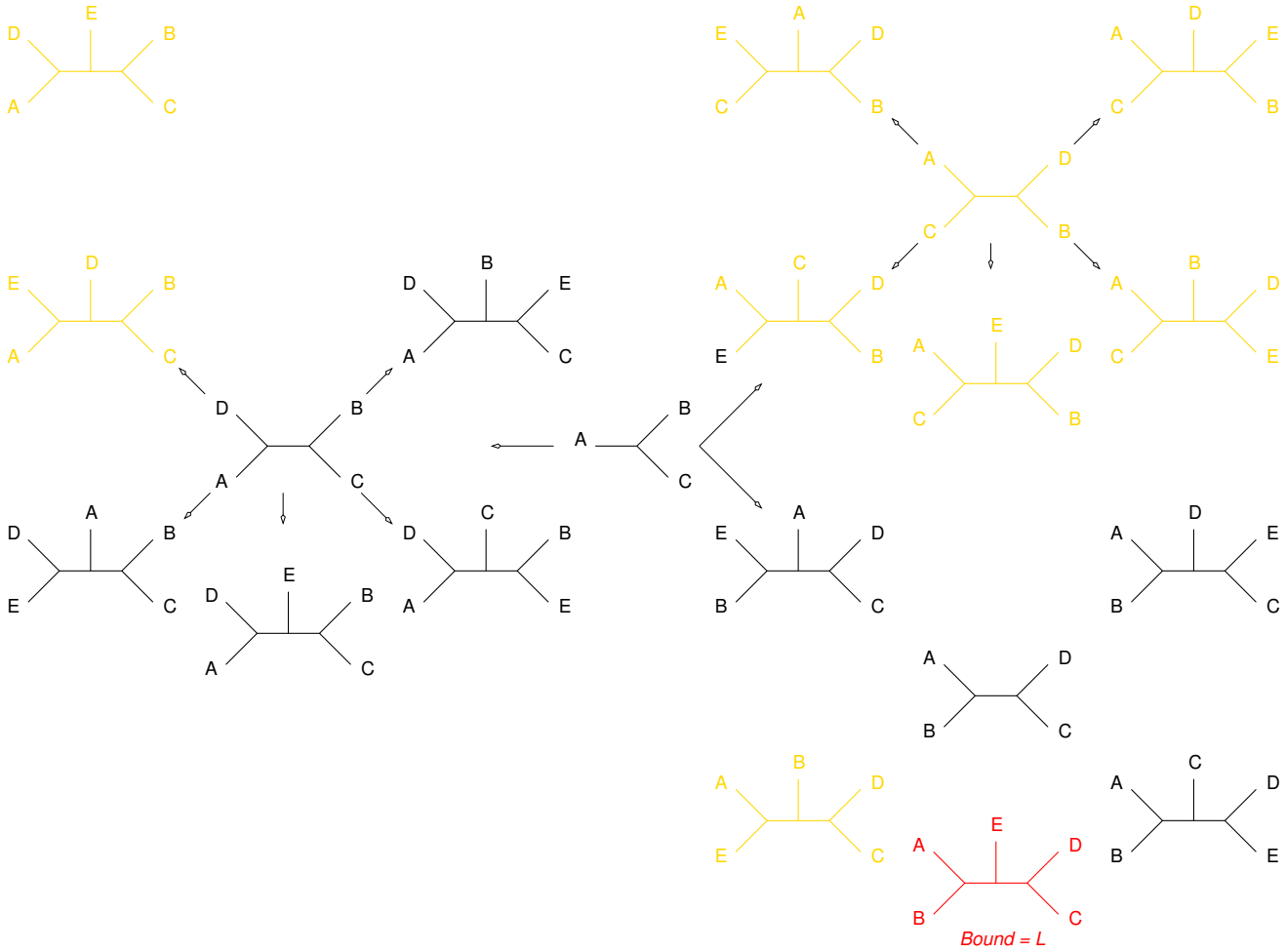
- **No** prescribed order to search the tree:
  - **Queue: breadth-first** search with branch and bound pruning;
  - **Stack: depth-first** search with branch and bound pruning;
  - **Priority queue: best-first** search with branch and bound pruning.

Branch and bound (version 1.0): 4 to 20 sequences

```
Let L, the minimum parsimony score far, be infinity.
Create two empty lists, open and solutions
Create an unrooted tree for three species and add it open
While open is not empty
  Remove the front element of the list and call it current
  Foreach tree t created by a sequential addition to current do
    If the minimum parsimony score of t is larger than L than discard t
      If the minimum parsimony score of t is is lower than  L
         If t has n leaves:
            clear solutions
            add t to solutions
            set L to the minimum parsimony score of t
          Else add t to the rear of open
      Else (equals case)
          If t has n leaves: add t to solutions
          Else add t to the rear of open
solutions is the list of all the solutions, their score is L.
```


Bound = L


Bound = L


Bound = L

Branch and bound (version 2.0)

How can you **improve** our algorithm?

Branch and bound (version 2.0)

How can you **improve** our algorithm?

- Estimating the **cost of extending a solution** (adding the remaining $n - k$ species to our tree, which already contains k sequences).

Branch and bound (version 2.0)

How can you **improve** our algorithm?

- Estimating the **cost of extending a solution** (adding the remaining $n - k$ species to our tree, which already contains k sequences).
- **How?**

Branch and bound (version 2.0)

How can you **improve** our algorithm?

- Estimating the **cost of extending a solution** (adding the remaining $n - k$ species to our tree, which already contains k sequences).
- **How?**
  - Each site (character) introducing new states (nucleotide) will increase the parsimony score.

Branch and bound (version 2.0)

	Sites (characters)					
Species	**1**	**2**	**3**	**4**	**5**	**6**
α	G	G	G	G	G	G
β	G	G	G	A	G	T
γ	G	G	A	T	A	G
δ	G	A	A	C	A	A
ϵ	G	G	T	C	A	C

Branch and bound (version 2.0)

```
Let L, the minimum parsimony score far, be infinity.
Create two empty lists, open and solutions
Create an unrooted tree for three species and add it open
While open is not empty
  Remove the front element of the list and call it current
  Foreach tree t created by a sequential addition to current do
      Let Lt be the minimum parsimony score of t + extension cost
      If Lt is larger than L than discard t
      If Lt is is lower than  L
         If t has n leaves:
            clear solutions
            add t to solutions
            set L to the minimum parsimony score of t
          Else add t to the rear of open
      Else (equals case)
          If t has n leaves: add t to solutions
          Else add t to the rear of open
solutions is the list of all the solutions, their score is L.
```

Branch and bound (version 3.0)

How can you **improve** our algorithm?

Branch and bound (version 3.0)

How can you **improve** our algorithm?

- Generate a realistic bound, using neighbour-joining, at the start of the algorithm.

Branch and bound (version 3.0)

```
Generate an initial tree T (using neighbour-joining method for instance)
Compute L the minimum parsimony score of T ( lowest score so far )
Create two empty lists, open and solutions
Create an unrooted tree for three species and add it open
While open is not empty
  Remove the front element of the list and call it current
  Foreach tree t created by a sequential addition to current do
      Let Lt be the minimum parsimony score of t + extension cost
      If Lt is larger than L than discard t
      If Lt is is lower than  L
         If t has n leaves:
            clear solutions
            add t to solutions
            set L to the minimum parsimony score of t
          Else add t to the rear of open
      Else (equals case)
          If t has n leaves: add t to solutions
          Else add t to the rear of open
solutions is the list of all the solutions, their score is L.
```

Branch and bound

Other ideas to improve the algorithm:

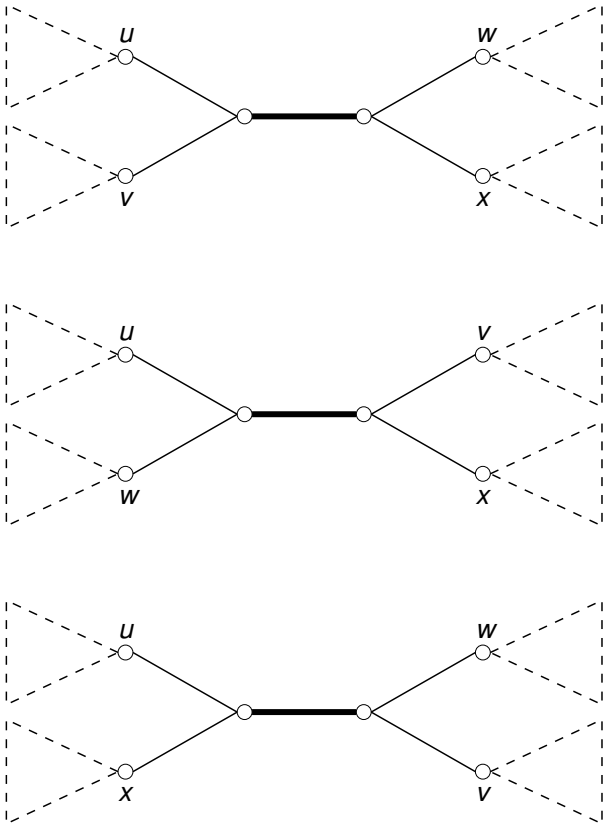
- Use a **priority queue** to store the partial solutions.
  - Thus always looking at the most promising solutions first.
- Derive a **tighter bound**:
  - Compatibility;
  - Zharkikh rules.

See [4].

Greedy algorithm

1. Generate an **initial topology** (using neighbour-joining, for instance);
2. Apply nearest **neighbour interchange** (NNI) transformations to all the internal edges;
3. Select the minimum parsimony tree;
4. Goto step 2.

$n > 20$: **Nearest-neighbour interchange** (NNI)



Other heuristics include: **subtree pruning and regrafting** (SPR) or **tree bisection and reconnection** (TBR).

Searching the **tree space**

- **Nearest-neighbour interchanges (NNI)**
  - Given an internal branch and its four connected nodes, u, v, w, x. NNI generates two novel solutions: one by exchanging the postions of v and w, the other by exchanging the positions of v and x.
    - Moves are very "local"
- **Subtree pruning and regrafting (SPR)**
  - Disconnects a subtree and reconnects that subtree in one of the branches of the remaining tree.
    - Wider search.
- **Tree bisection and reconnection**
  - Remove one branch, thus creating two subtrees. Consider all possible trees that are created by connecting one branch of the first subtree to another branch of the second subtree.

Discussion

- What are the **drawbacks** of greedy approaches?

Discussion

- What are the **drawbacks** of greedy approaches?
  - Finds a **local** optimum!

Remarks: distance-based vs character-based

- **Distance-based** methods compute the **pairwise sequence** distances **1) directly**, **2)** in isolation, **3)** before inferring the tree topology

Remarks: distance-based vs character-based

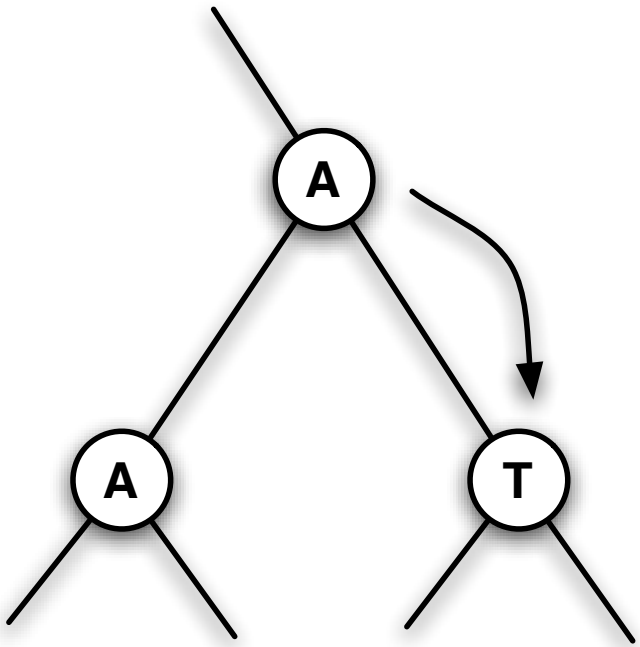
- **Distance-based** methods compute the **pairwise sequence** distances **1) directly**, **2)** in isolation, **3)** before inferring the tree topology
- Instead, for **character-based** methods, **1)** extant sequences are **never compared directly 2)** the pairwise distances depend on the reconstructed ancestral sequences, and **3)** this process (solving the small phylogeny problem) takes all the sequences into account

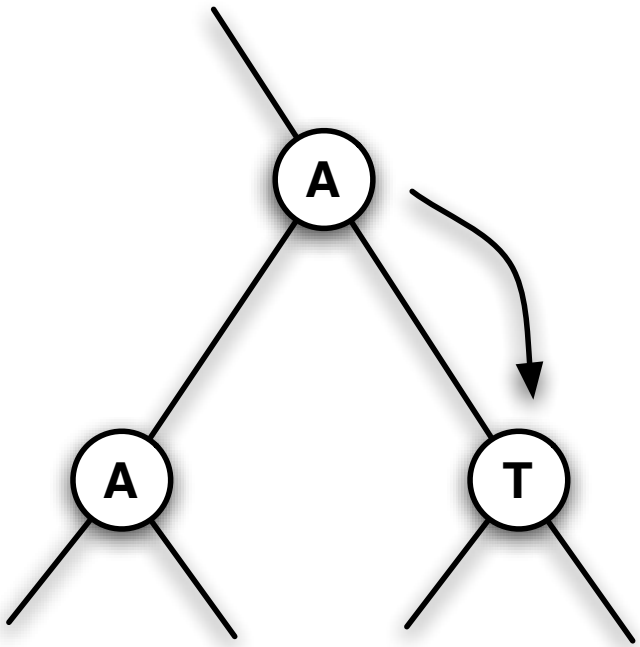
Remarks

- The particular methods that were presented are **not** modelling the base substitutions accurately.

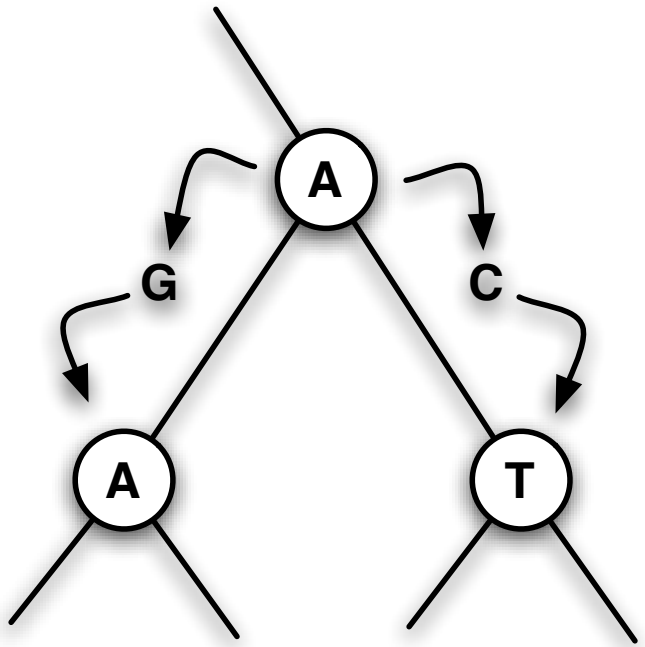
Remarks

- The particular methods that were presented are **not** modelling the base substitutions accurately.
- Specifically, these methods are ignoring the fact that **multiple substitutions** (for a given site) are likely to occur in any given branch of the tree (time interval).


A
A
T



vs



III. **Maximum likelihood** methods

Informal discussion!

III. **Maximum likelihood** methods

- $P(D|\Theta)$ denotes the **probability** of the **data** given some **model** Θ (set of parameters, such as tree topology, branch length, evolutionary model...).
- Let $L(\Theta) = P(D|\Theta)$ be the **likelihood function**.
- The **maximum likelihood estimate** is the value of Θ that maximizes $L(\Theta)$.

III. **Maximum likelihood** methods

- Let $L(\Theta)$ be the **likelihood** of a phylogenetic tree. $L(\Theta)$ is defined as the probability of the data (generally sequences) for a given tree (topology, branch length, evolutionary model), $P(\text{observed sequences}|\text{tree})$.

III. **Maximum likelihood** methods

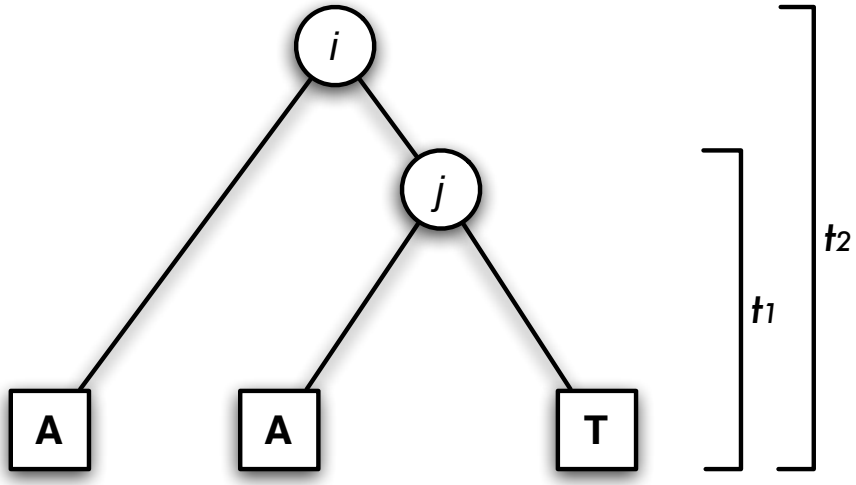
- Let $L(\Theta)$ be the **likelihood** of a phylogenetic tree. $L(\Theta)$ is defined as the probability of the data (generally sequences) for a given tree (topology, branch length, evolutionary model), $P(\text{observed sequences}|\text{tree})$.
- A **maximum likelihood** approach finds a tree, amongst all possible trees, with the largest value of $L(\Theta)$. Such tree explains best the data.

[See Felsenstein 2004, pages 251–253.]

Assumptions that are generally made:

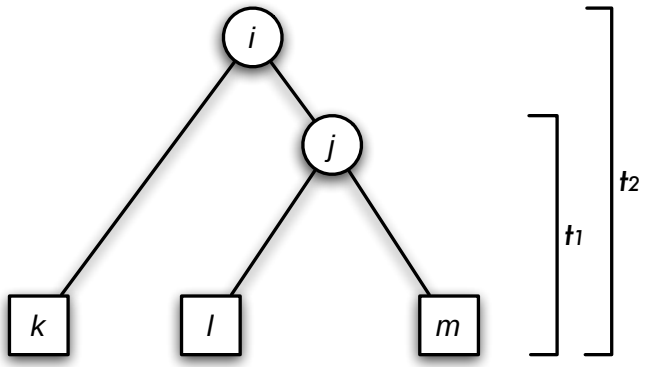
1. **Sites** are independent
2. **Lineages** are independent

Probability of a tree



$$\sum_i \sum_j p_i q_{iA}(t_2) q_{ij}(t_2 - t_1) q_{jA}(t_1) q_{jT}(t_1)$$

Probability of a tree (cont.)



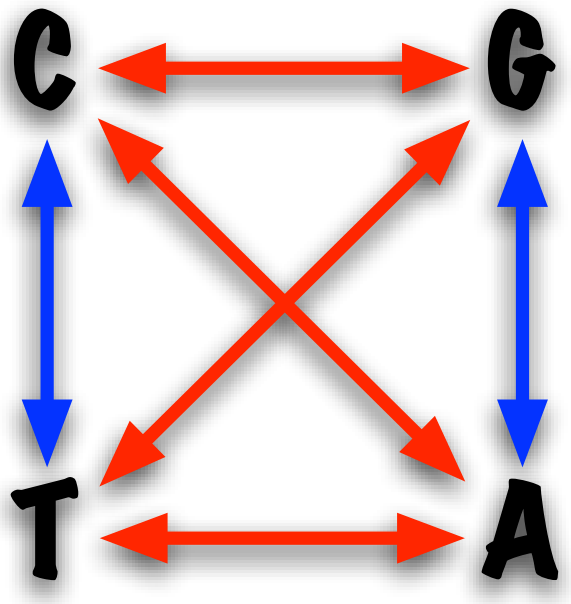
$$\sum_i \sum_j p_i q_{ik}(t_2) q_{ij}(t_2 - t_1) q_{jl}(t_1) q_{jm}(t_1)$$

where k, l, m are nucleotide types found at the given sequence position in the 3 organisms under study. Assuming that the positions (sites) are independent one from another (are evolving independently), the probability of the tree would be the product over all site probabilities.

Probability of a tree (cont.)

The $q_{ij}(t)$ terms give the probability of finding the nucleotide type j at a given site knowing that its ancestor had the nucleotide type i at the same position at time t (earlier).
Examples of substitution schemes modeling multiple substitutions for a given time interval include Jukes-Cantor one-parameter model and Kimura's two-parameter model.

Probability of a tree: model of evolution



Transition rate: blue and **transversion** rate: red

Probability of a tree: model of evolution

JC69: **Jukes** and **Cantor 1969**; bases are equiprobable; transition rate = transversion rate

Probability of a tree: model of evolution

JC69: **Jukes** and **Cantor 1969**; bases are equiprobable; transition rate $=$ transversion rate

K80: **Kimura 1980**; bases are equiprobable; transition rate $\neq$ transversion rate

Probability of a tree: model of evolution

JC69: **Jukes** and **Cantor 1969**; bases are equiprobable; transition rate = transversion rate

K80: **Kimura 1980**; bases are equiprobable; transition rate $\neq$ transversion rate

F81: **Felsenstein 1981**; variable base composition; transition rate = transversion rate

Probability of a tree: model of evolution

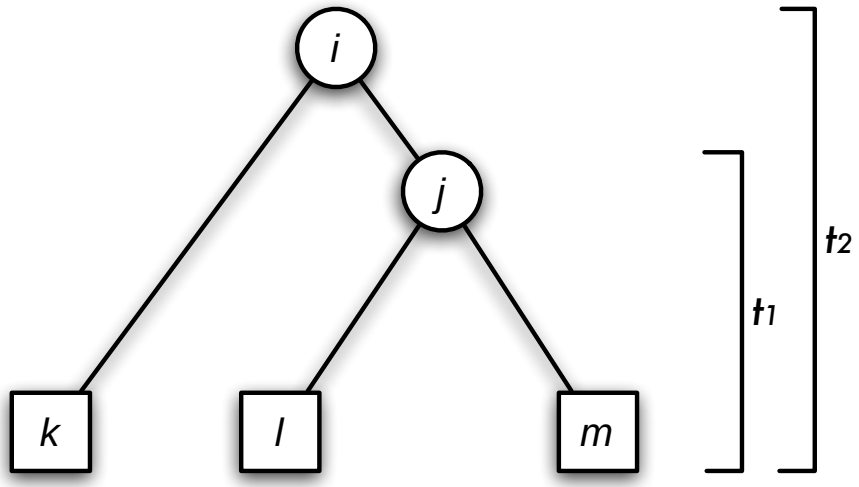
JC69: **Jukes** and **Cantor 1969**; bases are equiprobable; transition rate = transversion rate

K80: **Kimura 1980**; bases are equiprobable; transition rate $\neq$ transversion rate

F81: **Felsenstein 1981**; variable base composition; transition rate = transversion rate

HKY85: **Hasegawa** *et al.* **1985**; variable base composition; transition rate $\neq$ transversion rate; variable transition and transversion rates

Jukes and Cantor 1969



$$p(j|i,t) = \begin{cases} \frac{1}{4}(1+3e^{-4\alpha t}) & \text{if } j = i \\ \frac{1}{4}(1-e^{-4\alpha t}) & \text{if } j \neq i \end{cases}$$

where α is the mutation rate parameter.

Probability of a tree: model of evolution

In addition to the base model, most methods allow for relaxations:

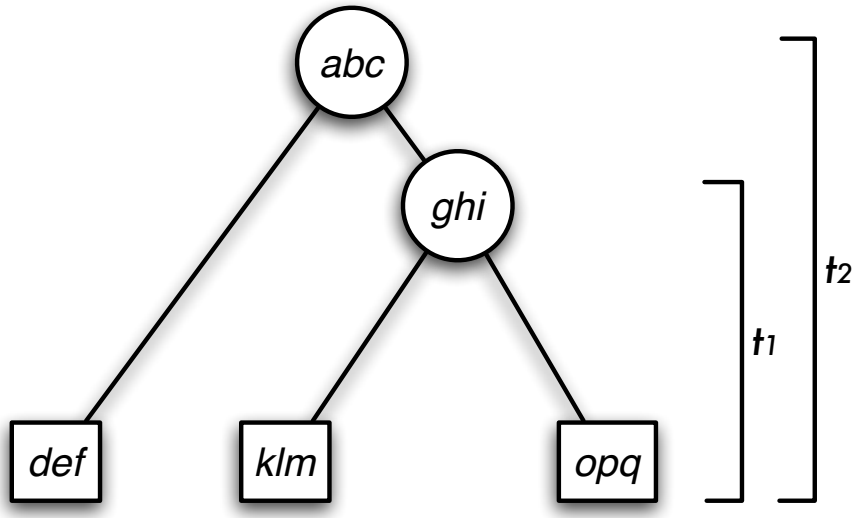
- Variable rates across positions ($+\Gamma$)

Probability of a tree: model of evolution

In addition to the base model, most methods allow for relaxations:

- Variable rates across positions ($+\Gamma$)
- Variable rates across lineages ($+\mathcal{I}$)

Probability of a tree: model of evolution



HKY85+$\Gamma + \mathcal{I}$ implies variable base composition, transition rate $\neq$ transversion rate, variable transition and transversion rates, that vary across sites and lineages.

Probability of a tree: model of evolution

- These models are also used to estimate pairwise distances for building phylogenies using distance-based approaches (e.g. Neighbour-joining).

Maximum likelihood methods

- Let L be the **likelihood** of a phylogenetic tree. L is defined as the probability of the data for a given tree, $P(\text{observed sequences}|\text{tree})$.
- A **maximum likelihood** approach finds a tree, amongst all possible trees, with the largest value of L. Such tree explains best the data.

Maximum likelihood methods

- Let L be the **likelihood** of a phylogenetic tree. L is defined as the probability of the data for a given tree, $P(\text{observed sequences}|\text{tree})$.
- A **maximum likelihood** approach finds a tree, amongst all possible trees, with the largest value of L. Such tree explains best the data.
- **Furthermore, the length of the branches are unknown and must be estimated as part of this process.**

Maximum likelihood methods

- Let L be the **likelihood** of a phylogenetic tree. L is defined as the probability of the data for a given tree, $P(\text{observed sequences}|\text{tree})$.
- A **maximum likelihood** approach finds a tree, amongst all possible trees, with the largest value of L. Such tree explains best the data.
- **Furthermore, the length of the branches are unknown and must be estimated as part of this process.**
- Finding an exact solution to this problem is impractical when the number of input sequences is large,

Maximum likelihood methods

- Let L be the **likelihood** of a phylogenetic tree. L is defined as the probability of the data for a given tree, $P(\text{observed sequences}|\text{tree})$.
- A **maximum likelihood** approach finds a tree, amongst all possible trees, with the largest value of L. Such tree explains best the data.
- **Furthermore, the length of the branches are unknown and must be estimated as part of this process.**
- Finding an exact solution to this problem is impractical when the number of input sequences is large, say 5 sequences/species.

Maximum likelihood methods

- Let L be the **likelihood** of a phylogenetic tree. L is defined as the probability of the data for a given tree, $P(\text{observed sequences}|\text{tree})$.
- A **maximum likelihood** approach finds a tree, amongst all possible trees, with the largest value of L. Such tree explains best the data.
- **Furthermore, the length of the branches are unknown and must be estimated as part of this process.**
- Finding an exact solution to this problem is impractical when the number of input sequences is large, say 5 sequences/species.
- Heuristic techniques have been developed to explore the tree space.

Maximum likelihood methods

- Generate an **initial tree topology** $\mathcal{T}$ (e.g. using NJ)
- Calculate its **likelihood** $\mathcal{L}$
- For a **fixed number of iterations**
  - From $\mathcal{T}$, generate **new trees** using NNI, SPR or TBR
  - For each new tree $\mathcal{T}'$, calculate its likelihood $\mathcal{L}'$
  - $\mathcal{L} = \mathcal{L}'$ and $\mathcal{T} = \mathcal{T}'$ if $\mathcal{L}' > \mathcal{L}$

Remarks: Parsimony vs Maximum Likelihood

- For a given tree topology, maximum parsimony considers all the reconstructions that lead to the same optimal score

Remarks: Parsimony vs Maximum Likelihood

- For a given tree topology, maximum parsimony considers all the reconstructions that lead to the same optimal score
- Maximum parsimony **under-estimates the number of evolutionary events** (because of the multiple substitutions along the branches of the tree)

Remarks: Parsimony vs Maximum Likelihood

- For a given tree topology, maximum parsimony considers all the reconstructions that lead to the same optimal score
- Maximum parsimony **under-estimates the number of evolutionary events** (because of the multiple substitutions along the branches of the tree)
- Maximum likelihood, through its evolutionary models, takes into account multiple substitutions, rate variations amongst sites and lineages, etc.

Remarks: Parsimony vs Maximum Likelihood

- For a given tree topology, maximum parsimony considers all the reconstructions that lead to the same optimal score
- Maximum parsimony **under-estimates the number of evolutionary events** (because of the multiple substitutions along the branches of the tree)
- Maximum likelihood, through its evolutionary models, takes into account multiple substitutions, rate variations amongst sites and lineages, etc.
- For a given tree topology, maximum likelihood considers all the reconstructions (and not only the most parsimonious ones)

Remarks: Parsimony vs Maximum Likelihood

- For a given tree topology, maximum parsimony considers all the reconstructions that lead to the same optimal score
- Maximum parsimony **under-estimates the number of evolutionary events** (because of the multiple substitutions along the branches of the tree)
- Maximum likelihood, through its evolutionary models, takes into account multiple substitutions, rate variations amongst sites and lineages, etc.
- For a given tree topology, maximum likelihood considers all the reconstructions (and not only the most parsimonious ones)
- This is the most time consuming approach of all three

Other issues: informative sites

- Intuitively, the sites (columns) of an alignment that contain a single nucleotide type (**invariant** sites) provide no useful information for building a phylogenetic tree using a character-based approach.

Other issues: informative sites

- Intuitively, the sites (columns) of an alignment that contain a single nucleotide type (**invariant** sites) provide no useful information for building a phylogenetic tree using a character-based approach.
- A site is **informative** if it allows to discriminate between trees, i.e. the minimum parsimony scores for at least two trees are different for that site, otherwise the site is **uninformative**.

Other issues: informative sites

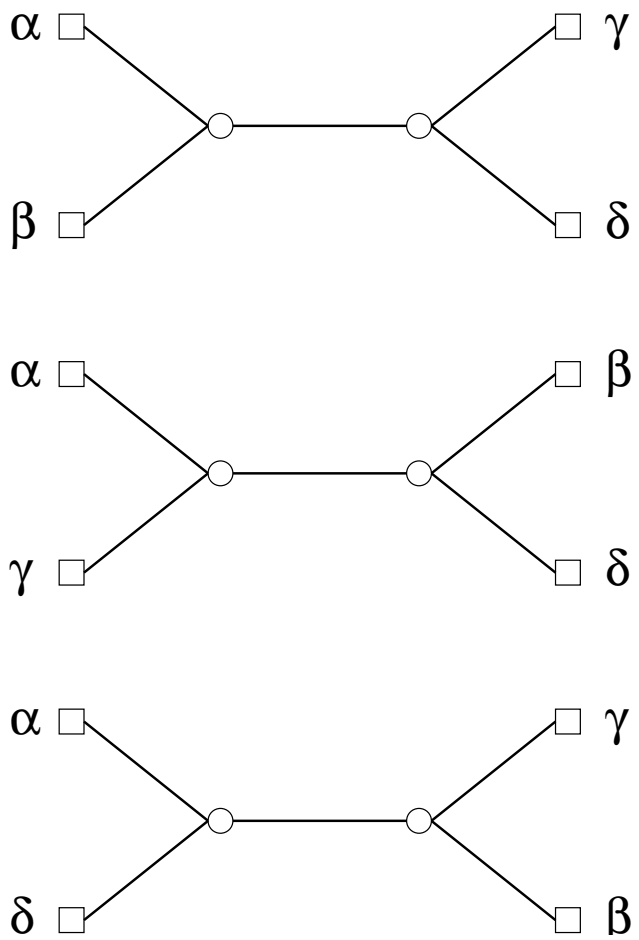
- Intuitively, the sites (columns) of an alignment that contain a single nucleotide type (**invariant** sites) provide no useful information for building a phylogenetic tree using a character-based approach.
- A site is **informative** if it allows to discriminate between trees, i.e. the minimum parsimony scores for at least two trees are different for that site, otherwise the site is **uninformative**.
- Clearly, invariant sites are uninformative.

Informative sites

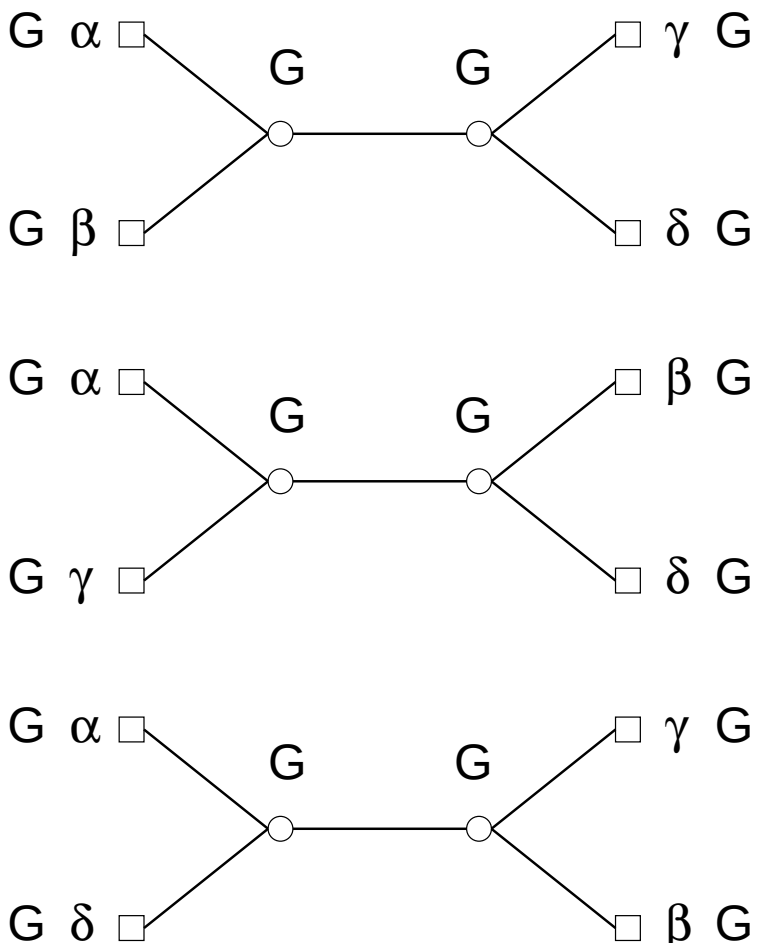
	Site					
Species	**1**	**2**	**3**	**4**	**5**	**6**
α	G	G	G	G	G	G
β	G	G	G	A	G	T
γ	G	G	A	T	A	G
δ	G	A	T	C	A	T

$\Rightarrow$ Adapted from [3, pages 99–101].

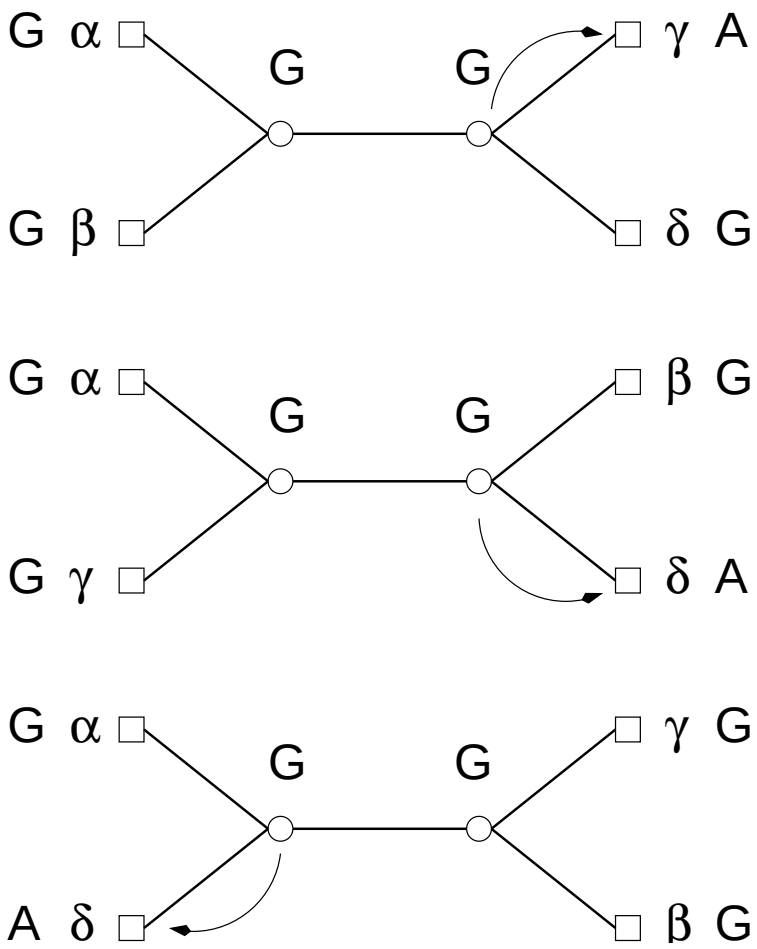
Given 4 species, there are 3 possible unrooted trees



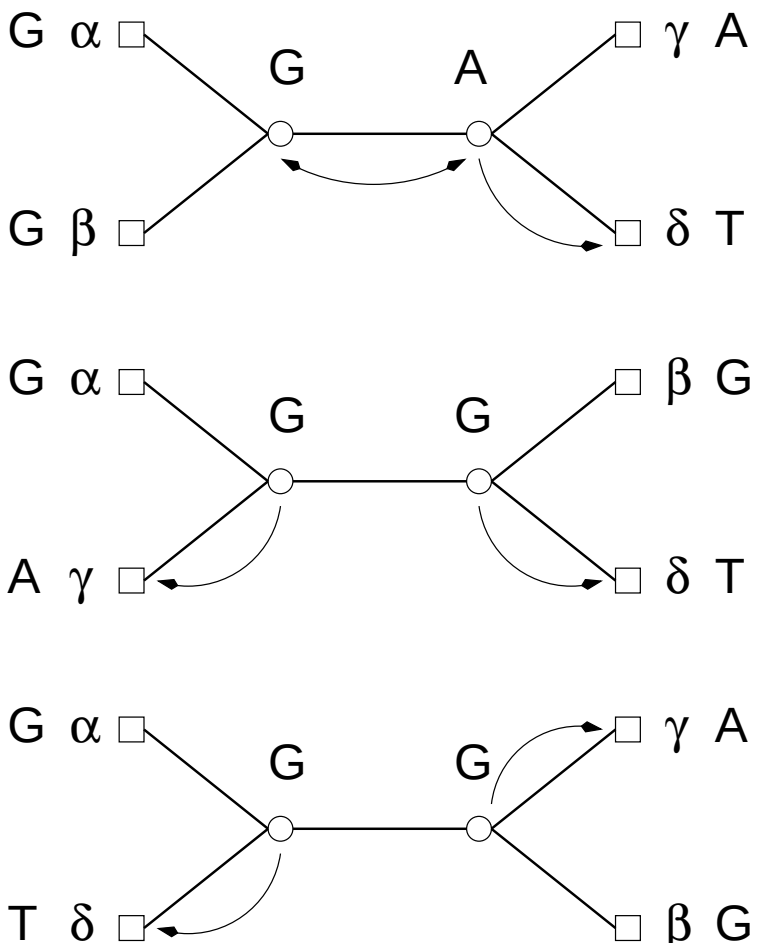
Site 1



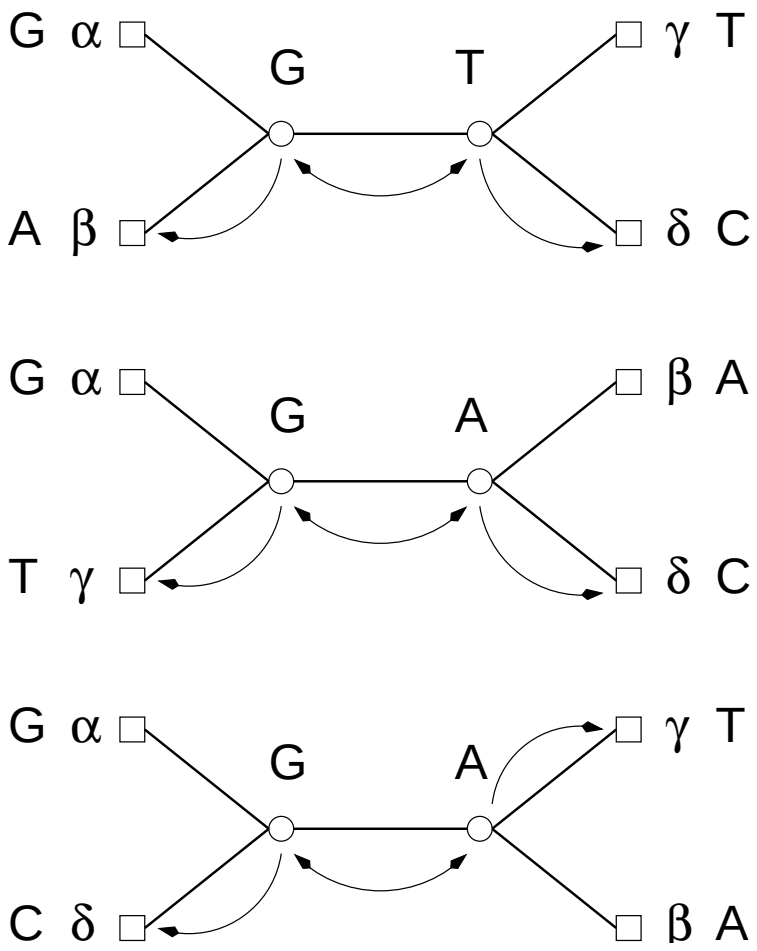
Site 2



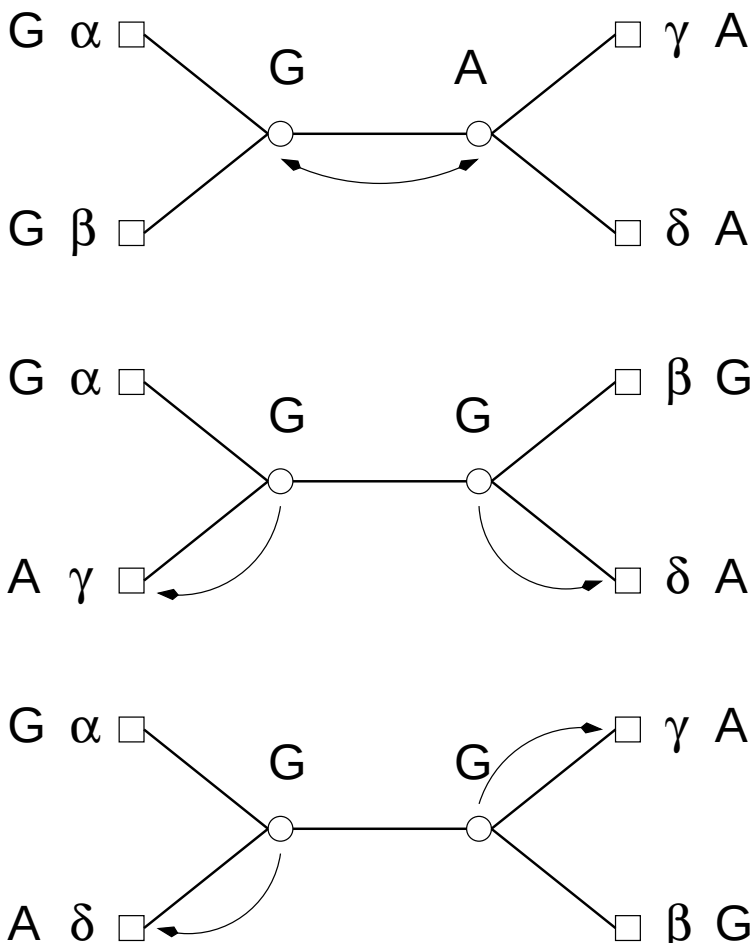
Site 3



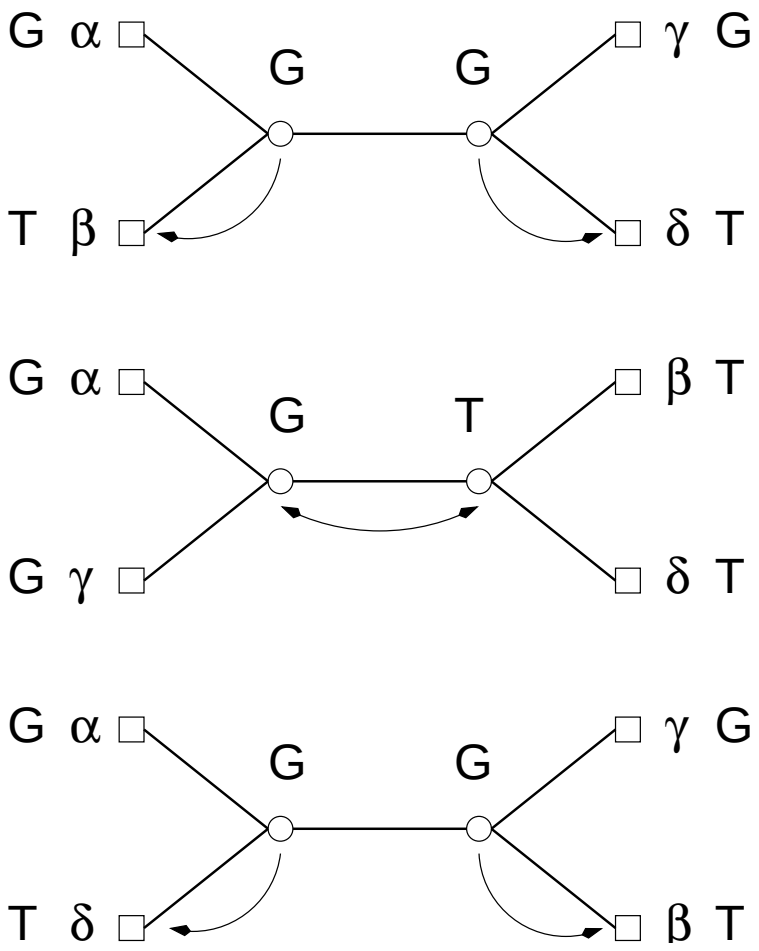
Site 4



Site 5



Site 6



Informative sites

- Fortunately, there is a simple rule to identify informative sites. There has be at least two types that are occur at least twice at that site.
- Uninformative sites are discarded prior to the inference of the tree.
- Notice that those sites are typically kept by distance-based methods. This partly explains why the methods are producing different results.

Further readings

- **Bayesian inference** of phylogenetic trees
- **Quartet** methods
- Evolutionary **networks** (as opposed to evolutionary trees)
- **Bootstraps**, **consensus**, **comparing trees**
- Matching interior nodes (taxonomic units) with paleontological information, so as to assign time to events

References

Wing-Kin Sung.
Algorithms in Bioinformatics: A Practical Introduction.
Chapman & Hall/CRC Mathematical and Computational Biology. Chapman and Hall/CRC, 1st edition edition, 2009.

Bernhard Haubold and Thomas Wiehe.
Introduction to computational biology: an evolutionary approach.
Birkhäuser Basel, 2006.

D. E. Krane and M. L. Raymer.
Fundamental Concepts of Bioinformatics.
Benjamin Cummings, 2003.

References (cont.)

P W Purdom Jr, P G Bradford, K Tamura, and S Kumar. Single column discrepancy and dynamic max-mini optimizations for quickly finding the most parsimonious evolutionary trees.
Bioinformatics (Oxford, England), 16(2):140–151, February 2000.

Pensez-y!

L'impression de ces notes n'est probablement pas nécessaire!