Melbourne Bioinformatics

BIOINFORMATICS + DATA SERVICES + INFRASTRUCTURE, FOR LIFE SCIENCES TODAY

THE UNIVERSITY OF MELBOURNE

COMP90014

Algorithms for Bioinformatics

Week 5B - Evolutionary Trees II

Evolutionary Trees II

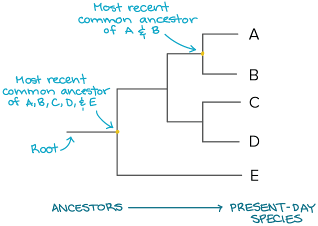
Recap

Distance vs Character methods

Evaluating trees (character methods)
- Fitch algorithm
- Sankoff algorithm

Building trees (character methods)
- Heuristics
- Reliability

Phylogenetics


Most recent common ancestor of A & B
Most recent common ancestor of A, B, C, D, & E
Root
A
B
C
D
E
ANCESTORS
PRESENT-DAY SPECIES

Taxonomy: the science of classifying organisms

Phylogenetics: describes the evolutionary relationship between species

Speciation: A population of organisms becomes separated.

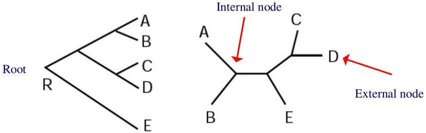
Over time, these evolve into separate species that do not cross-breed.

Robert Bear via Khan Academy

How many different trees can we construct with *n* sequences?

Sequences	Unrooted trees
3	1
4	3
5	15
10	$> 2\,000\,000$

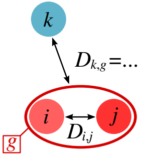
Unrooted: $\prod_{i=3}^{n} (2i - 5)$



Rooted: (more or less?)

- enumerating all possible trees to find the best one is not feasible
- optimisation approach:
  - which tree minimises number of changes needed to explain data (parsimony)?
  - which minimises the distance between taxa?

UPGMA algorithm



Consider a distance matrix D, and n groups containing one item/leaf each:

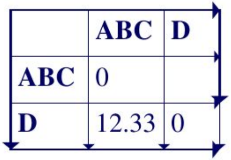
1. Choose the i and j with the smallest D_{ij}
2. Create a new group ij
3. Connect i and j to a new node in the tree that correspond to the new group
4. Set the branch length to $\frac{D_{ij}}{2}$ (ultrametric)
5. Calculate the distance between the group and all existing groups (n_i = number of elements):

$$D_{(ij),k} = \left(\frac{n_i}{n_i + n_j}\right)D_{ik} + \left(\frac{n_j}{n_i + n_j}\right)D_{jk}$$

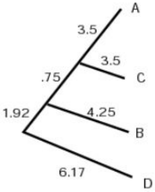
6. Replace the i and j columns with the new group
7. If there is only one item left stop, otherwise go to 1

Thibaut Jombart, Introduction to phylogenetics

Example



	ABC	D
ABC	0	
D	12.33	0



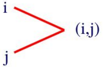
- UPGMA assumes that the rates of evolution are the same among different items
- We don't use this method for phylogenetic tree reconstruction (unless we believe the assumption...!)

UPGMA algorithm

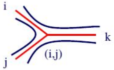
1. Choose smallest D_{ij}
2. Create a new group ij
3. Set the branch length to $\frac{D_{ij}}{2}$
4. Update distance matrix
5. Replace the i and j columns with the new group

Neighbour joining algorithm

$$u_i = \sum_{j:j \neq i}^{n} \frac{D_{ij}}{n-2}$$



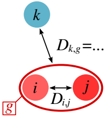
$$v_i = \frac{1}{2} D_{ij} + \frac{1}{2}(u_i - u_j)$$
$$v_j = \frac{1}{2} D_{ij} + \frac{1}{2}(u_j - u_i)$$



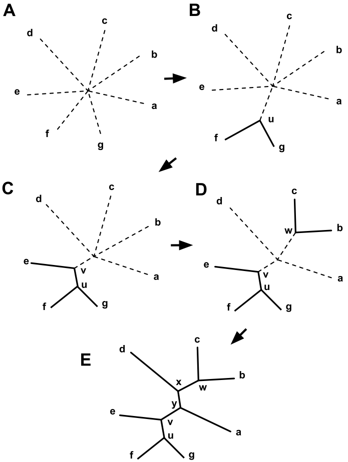
$$D_{(ij)k} = \frac{D_{ik} + D_{jk} - D_{ij}}{2}$$

Consider a distance matrix D:

1. Calculate the “average” distance to other nodes/clusters for each leaf
2. Choose i and j to minimize $D_{ij} - u_i - u_j$
   (Nodes that are close to each other, and far from everything else)
3. Join i and j to create a new node (i, j) and calculate the new branch lengths
4. Compute distance between leaves and the new group
5. Replace the i and j leaves with the new node (i, j)
6. Continue until two nodes remain



Neighbour joining, graphically



Tomfy via WikiMedia Commons

1. begin with a star tree
2. minimise $D_{ij} - u_i - u_j$
3. resolve pairs
4. update distance matrices
5. go to 2

🌲 neighbour joining does not assume all sequences evolve at the same rate

Evolutionary Trees II

Recap

Distance vs Character methods

Evaluating trees (character methods)
- Fitch algorithm
- Sankoff algorithm

Building trees (character methods)
- Heuristics
- Reliability

Phylogeny reconstruction algorithms

Two types of reconstruction:

Distance-based

- A tree is built based on the distance between items
- Closer taxa should be more evolutionarily related

- UPGMA
- neighbour-joining

Character-based

- Every taxon is described by a number of characters
  e.g. number of fingers, protein sequence.
- each has a finite number of states.
- Goal: build the tree that best explains the character matrix
  - Optimise a objective function

- Maximum Parsimony
- Maximum Likelihood

Distance-based methods

Advantages

- simple
- flexible
- fast and scalable

Limitations

- sensitive to distance method
- evolutionary rates are not estimated
- no measure of uncertainty for the tree obtained

Phylogeny reconstruction algorithms

这类算法不直接使用物种之间的距离来构建树，而是根据物种的多种特征来描述每一个类群（taxon）。包括物种的形态特征，如手指数量，或分子特征，如蛋白质序列。

Two types of reconstruction:

Distance-based

- A tree is built based on the distance between items
- Closer taxa should be more evolutionarily related

- UPGMA
- neighbour-joining

Character-based

- Every taxon is described by a number of characters
  e.g. number of fingers, protein sequence.
- each has a finite number of states.
- Goal: build the tree that best explains the character matrix
  - Optimise a objective function

- Maximum Parsimony
- Maximum Likelihood

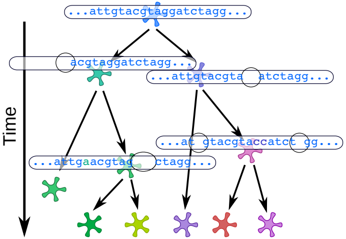
最大简约法（Maximum Parsimony），尽量选择最简单的解释方案，即产生最少进化改变的树。

最大似然法（Maximum Likelihood），根据特定的进化模型，找到使得观察到的数据出现概率最大的树。

简约性方法

Parsimony methods

L(T)表示树T所需的最少替代（substitutions）次数来解释所有的数据。



该树的L(T)最小化，即通过最少的替换次数就能解释这些序列之间的差异。

Thibaut Jombart, Introduction to phylogenetics

Parsimony: simpler is better. See Occam's razor.

- *i.e.* build the tree with the fewest point mutations

Parsimony length/score

- $L(T)$ is the minimum number of substitutions required to explain tree T.

Assumption

- characters are independent
- so are the changes in different columns (species)

Parsimony problem

- compute a phylogenetic tree T for a set of sequences that minimizes $L(T)$

Computational problems

Small parsimony

- given a tree T with each leaf labeled by a sequence, ***calculate the parsimony length*** $L(T)$ and the corresponding labeling of internal nodes
- evaluating a tree is easy:
  - Fitch's algorithm
  - Sankoff's algorithm

Large (maximum) parsimony

- given the character matrix M, ***compute the most parsimonious tree for*** M
- NP-hard: enumerating trees is intractable
- alternatives:
  - heuristics
  - branch and bound

Evolutionary Trees II

Recap

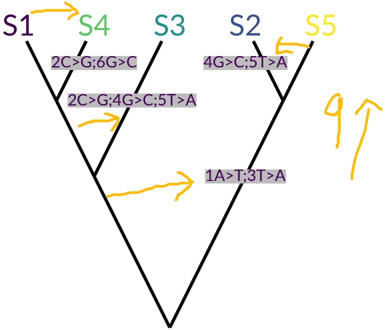
Distance vs Character methods

<u>Evaluating trees (character methods)</u>
- Fitch algorithm
- Sankoff algorithm

Building trees (character methods)
- Heuristics
- Reliability

Evaluating trees

S1	ACTGTG
S2	TCACAG
S3	AGTCAG
S4	AGTGTC
S5	TCAGTG



- Label internal nodes, *e.g.* Hamming distance (number of changes)

A candidate tree:

- $L(T) = 9$ changes
- How can we make this tree more parsimonious?

A better tree:

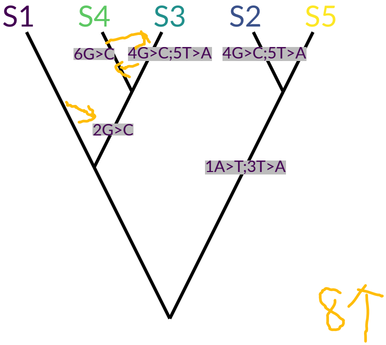
- $L(T) = 8$ changes

An equally good tree:

- $L(T) = 8$ changes

Evaluating trees

S1	ACTGTG
S2	TCACAG
S3	AGTCAG
S4	AGTGTC
S5	TCAGTG



- Label internal nodes, *e.g.* Hamming distance (number of changes)

A candidate tree:

- $L(T) = 9$ changes
- How can we make this tree more parsimonious?

A better tree:

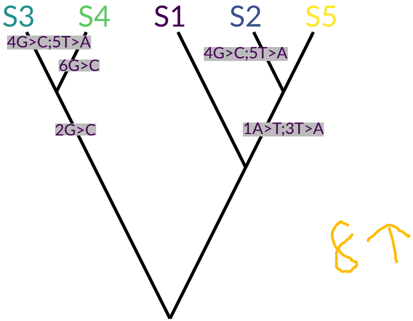
- $L(T) = 8$ changes

An equally good tree:

- $L(T) = 8$ changes

Evaluating trees

S1	ACTGTG
S2	TCACAG
S3	AGTCAG
S4	AGTGTC
S5	TCAGTG



- Label internal nodes, *e.g.* Hamming distance (number of changes)

A candidate tree:

- $L(T) = 9$ changes
- How can we make this tree more parsimonious?

A better tree:

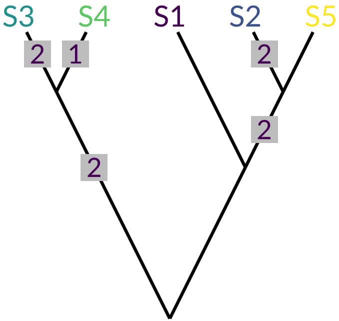
- $L(T) = 8$ changes

An equally good tree:

- $L(T) = 8$ changes

Evaluating trees

S1	ACTGTG
S2	TCACAG
S3	AGTCAG
S4	AGTGTC
S5	TCAGTG



- Label internal nodes, *e.g.* Hamming distance (number of changes)

A candidate tree:

- $L(T) = 9$ changes
- How can we make this tree more parsimonious?

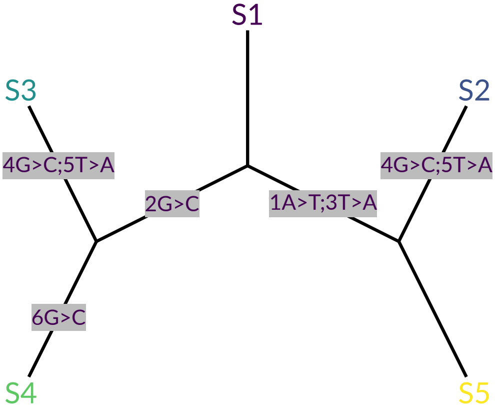
A better tree:

- $L(T) = 8$ changes

An equally good tree:

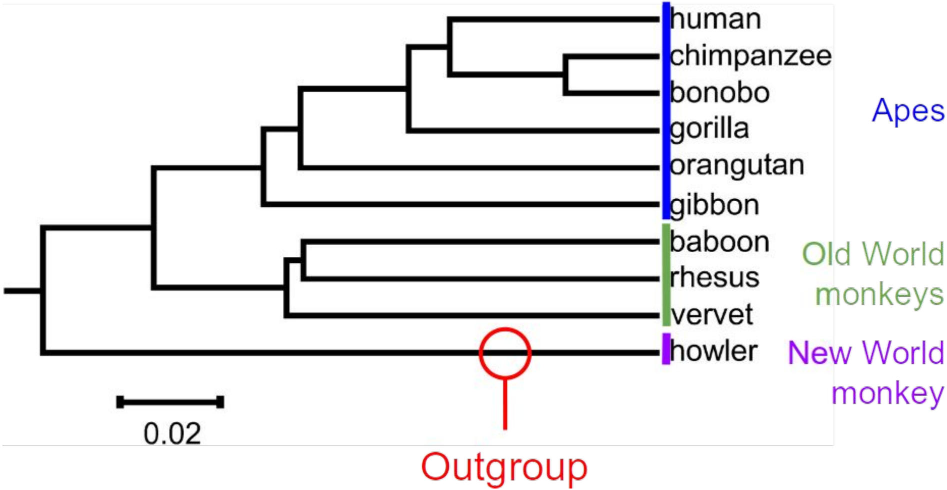
- $L(T) = 8$ changes

Rooted or unrooted trees



- 🌲 both trees with $L(T) = 8$ are the same when unrooted
- 🌲 the root can be placed on any branch

Rooting a tree: outgroups



Kalinowski *et al.*, 2016. 10.1534/genetics.105.047977.

Small parsimony: computational problems

Small parsimony

- given a tree T, calculate $L(T)$
- for small trees we can calculate by hand
- impractical for larger trees with many leaves

Algorithmically:

- iterate over positions in the alignment
- at each position, find internal nodes that require a mutation to explain the data found in the children

Fitch algorithm

- dynamic programming
- compute parsimony score for a column of the sequence alignment
- repeat the process for each column
- substitutions have the same cost

Sankoff algorithm

- dynamic programming
- allows us to calculate the cost of changes in a given tree

Evolutionary Trees II

Recap

Distance vs Character methods

Evaluating trees (character methods)
- Fitch algorithm
- Sankoff algorithm

Building trees (character methods)
- Heuristics
- Reliability

Fitch algorithm

Input: a phylogenetic tree T

- n nodes
- a single character column c with a set A of k possible values
- denote the value of the character for node v by v_c.

Step 1: Assign to each node v a set $S_v \in A$ as follows:

- For each leaf v: $S_v = \{v_c\}$
- For any internal node v, with children u, w:

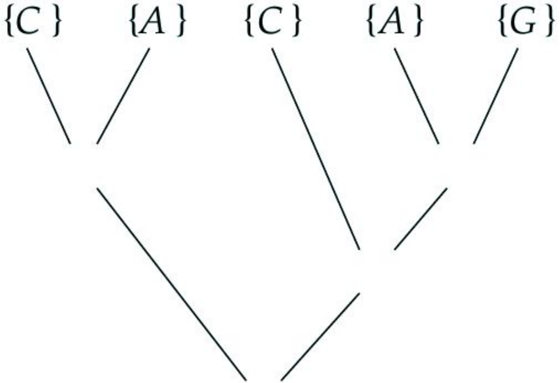
$$S_v = \{S_u \cap S_w \text{ if } S_u \cap S_w \neq 0 \\ S_u \cup S_w \text{ otherwise}\}$$

- Compute S_v with postorder tree traversal – starting with the leaves

Step 2: Traverse tree in preorder, to determine the value v_c to assign to each internal node v

- The number of changes in this tree is equal to the number of times $S_u \cap S_w = 0$

Fitch algorithm: example 1



For each leaf *v*:

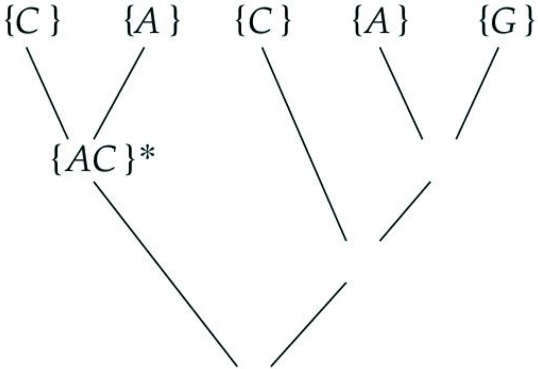
$$S_v = \{v_c\}$$

For any internal node *v*:

$$S_v = \{S_u \cap S_w \text{ if } S_u \cap S_w \neq 0$$
$$S_u \cup S_w \text{ otherwise}\}$$

- $L(T) = 3$
- Repeat the process for each column
- Changes have the same cost

Fitch algorithm: example 1



For each leaf *v*:

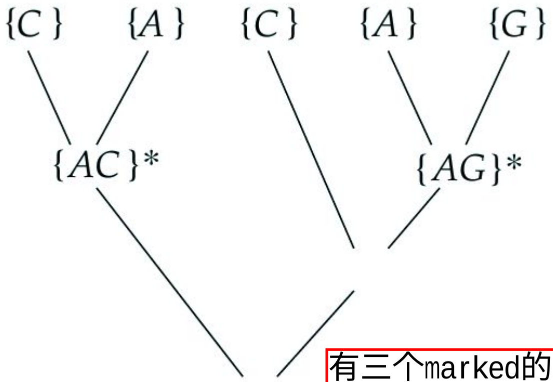
$$S_v = \{v_c\}$$

For any internal node *v*:

$$S_v = \{S_u \cap S_w \text{ if } S_u \cap S_w \neq 0$$
$$S_u \cup S_w \text{ otherwise}\}$$

- $L(T) = 3$
- Repeat the process for each column
- Changes have the same cost

Fitch algorithm: example 1



有三个marked的union set 所以有3个mutation needed to explain the data

For each leaf v:

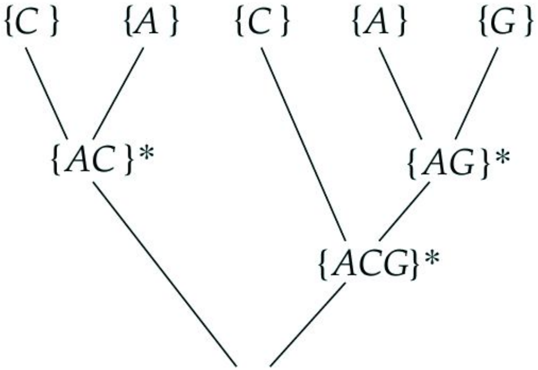
$$S_v = \{v_c\}$$

For any internal node v:

$$S_v = \{S_u \cap S_w \text{ if } S_u \cap S_w \neq 0$$
$$S_u \cup S_w \text{ otherwise}\}$$

- 🌲 $L(T) = 3$
- 🌲 Repeat the process for each column
- 🌲 Changes have the same cost

Fitch algorithm: example 1



For each leaf *v*:

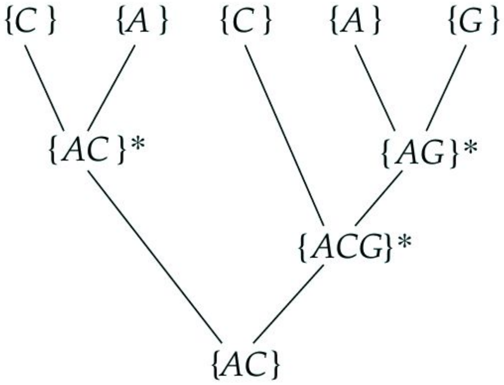
$$S_v = \{v_c\}$$

For any internal node *v*:

$$S_v = \{S_u \cap S_w \text{ if } S_u \cap S_w \neq 0$$
$$S_u \cup S_w \text{ otherwise}\}$$

- $L(T) = 3$
- Repeat the process for each column
- Changes have the same cost

Fitch algorithm: example 1



For each leaf v:

$$S_v = \{v_c\}$$

For any internal node v:

$$S_v = \{S_u \cap S_w \text{ if } S_u \cap S_w \neq 0$$

$$S_u \cup S_w \text{ otherwise}\}$$

- $L(T) = 3$
- Repeat the process for each column
- Changes have the same cost

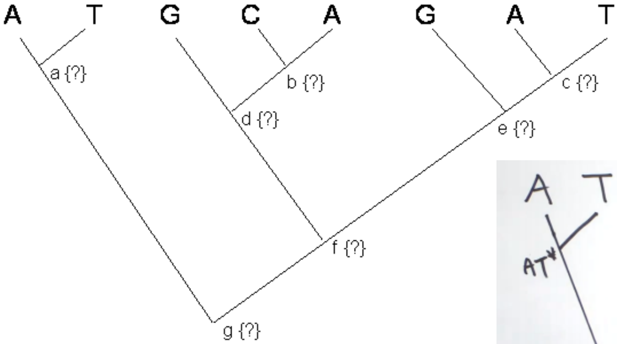
Fitch algorithm: example 2

For each leaf v:

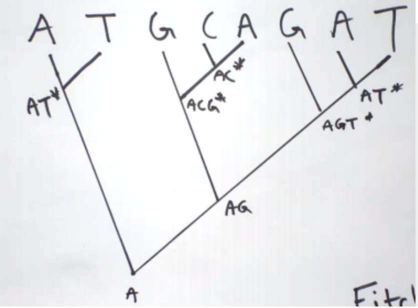
$$S_v = \{v_c\}$$

For any internal node v:

$$S_v = \{S_u \cap S_w \text{ if } S_u \cap S_w \neq 0$$
$$S_u \cup S_w \text{ otherwise}\}$$



L(T) = 5



```
a{?}
b{?}
d{?}
c{?}
e{?}
f{?}
g{?}
----------

L(T) = ?
```

Evolutionary Trees II

Recap

Distance vs Character methods

Evaluating trees (character methods)
- Fitch algorithm
- Sankoff algorithm

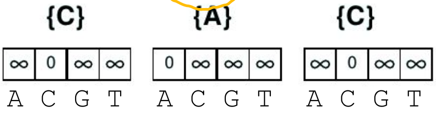
Building trees (character methods)
- Heuristics
- Reliability

Sankoff algorithm

Count the smallest number of possible (weighted) changes needed on a given tree

Cost for the leaves

- 0 for the observed letter
- infinity otherwise



{C}

A	C	G	T
∞	0	∞	∞

{A}

A	C	G	T
0	∞	∞	∞

{C}

A	C	G	T
∞	0	∞	∞

Calculate costs for internal nodes

- for each node, compute the minimum cost S_a for each character i to occur at that node

$$S_a(i) = \min[c_{ij} + S_L(j)] \\ + \min[c_{ik} + S_R(k)]$$

- L and R are left and right children nodes
- c_{ij} is the cost for changing from state i to j

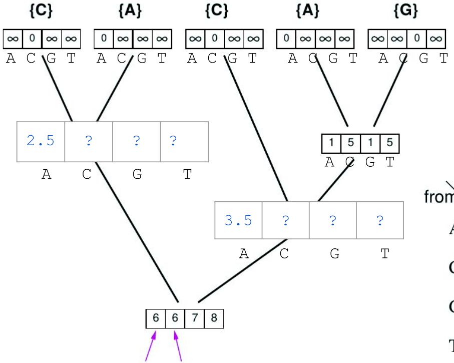
Use a cost matrix

- we used a fixed cost for Fitch's algorithm
- for Sankoff, we use a cost matrix

编辑距离：编辑距离是衡量两条序列相似性的一个指标，它代表将一条序列变成另一条序列所需的最少编辑操作数（如插入、删除或替换）。
代价矩阵：代价矩阵是一个表格，描述了各种编辑操作的代价。
Sankoff算法用于计算两条或多条RNA序列及其二级结构之间的最小编辑距离，并输出最优比对结果。这一过程是通过动态规划完成的，其中每个子问题都是比对序列的一个小段，并考虑其二级结构。

Sankoff example

A=min[0, · , · , ·] + min[· , · , 1, ·]=0+1 = 1



cost matrix:

from \ to	A	C	G	T
A	0	2.5	1	2.5
C	2.5	0	2.5	1
G	1	2.5	0	2.5
T	2.5	1	2.5	0

S_i?

$$i = A$$
$$j = A, C, G, T$$
$$k = A, C, G, T$$

$$S_a(i) = \min[c_{ij} + S_L(j)] + \min[c_{ik} + S_R(k)]$$

🌲 Limitation: implicitly assumes that rate of change along branches is similar

L(T) = 6

Evolutionary Trees II

Recap

Distance vs Character methods

Evaluating trees (character methods)
- Fitch algorithm
- Sankoff algorithm

Building trees (character methods)
- Heuristics
- Reliability

Parsimony:· computational problems

这张幻灯片讨论了在系统发育分析中寻找最简约树（即反映物种关系最少变化的树）时遇到的计算问题。

- we know how to score a tree for parsimony (***small parsimony***)
- how can we find the best tree? (***large/maximum parsimony***)
  - optimization problem
- enumerating trees is unfeasible
  - $O(n!)$: factorial growth with the number of leaves (*e.g.* sequences)
  - not feasible to score all of them
  - heuristic approach
  - tree searching methods

Sequences	Unrooted trees
3	1
4	3
5	15
10	$> 2\,000\,000$

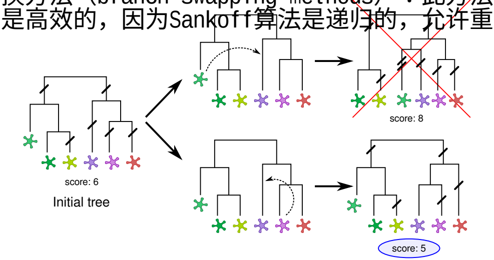
枚举树的不可行性：理论上，我们可以通过计算所有可能的树并为每棵树打分来找到最简约的树。但是，这在计算上是不可行的，因为可能的树的数量随序列的数量呈阶乘增长（O(n!)）。这意味着即使是少量的序列也会产生大量可能的树，例如10个序列就有超过200万种不同的未根树。

不可行的评分所有树：由于可能的树的数量如此之多，因此不可能为所有可能的树打分以找到得分最高的那棵

Exploring tree space

顺序/分步添加（sequential/stepwise addition）：这个过程涉及逐步添加物种或序列来构建系统发育树，并尝试找到最佳解。

树枝交换方法（branch swapping methods）：此方法包括通过断开和重新连接树枝来重新排列树。这个过程是高效的，因为Sankoff算法是递归的，允许重新评分一个树结构，而不需要重新计算整个



Exact methods

- exhaustive search
- branch and bound algorithms
  - reduce search space
  - eliminate candidate solutions that will not reach an optimal solution

Heuristics

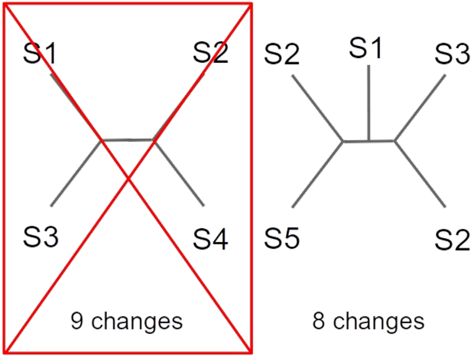
- sequential/stepwise addition
- branch swapping methods
  - we can rearrange trees by breaking and reattaching branches
  - efficient to re-score because Sankoff algorithm is recursive

穷举搜索（exhaustive search）：这是一种尝试所有可能解的方法，以找到最佳系统发育树。

分支限界算法（branch and bound algorithms）：这个算法通过减少搜索空间，消除那些不会达到最优解的候选解，提高了搜索的效率。

Thibaut Jombart, Introduction to phylogenetics

Branch and bound



- this was our best five-tip tree $L(T) = 8$ (8 changes)
- once we find that, we don’t have to look at trees based on the four-tip tree with 9 changes
- reduces search space
- always finds the optimal

Evolutionary Trees II

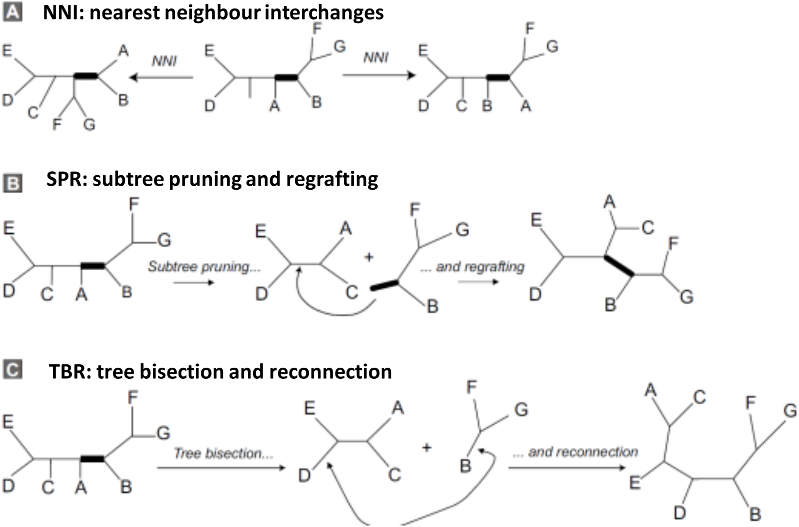
Recap

Distance vs Character methods

Evaluating trees (character methods)
- Fitch algorithm
- Sankoff algorithm

Building trees (character methods)
- Heuristics
- Reliability

Heuristic: branch Swapping



A NNI: nearest neighbour interchanges

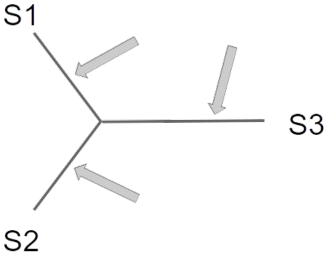
B SPR: subtree pruning and regrafting

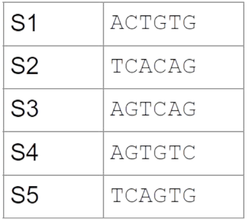
C TBR: tree bisection and reconnection

Kai Müller via WWU Münster

Heuristic: sequential addition

- assume the tree is unrooted for simplicity
- we can add S4 in three places

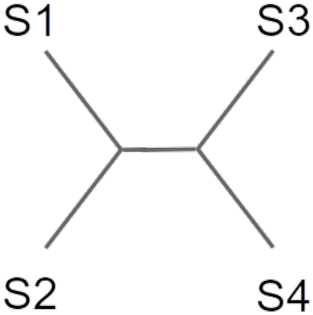




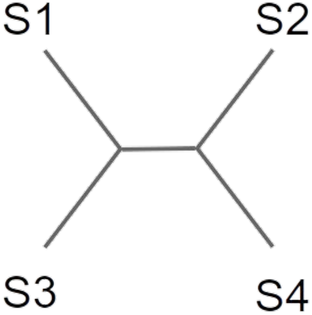
S1	ACTGTG
S2	TCACAG
S3	AGTCAG
S4	AGTGTC
S5	TCAGTG

Heuristic: sequential addition

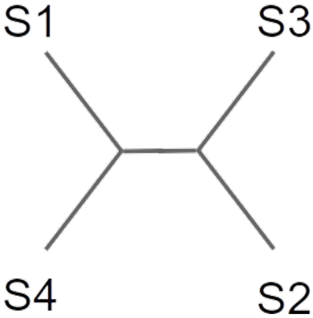
使用启发式方法中的顺序添加（sequential addition）来构建系统发育树的例子。在顺序添加方法中，我们逐步构建树，每次添加一个物种（或序列），并计算每次添加后树的变化数。目的是找到变化数最少的树，因为在许多系统发育分析中，变化数最少的树被认为是最符合数据的树。



8 changes



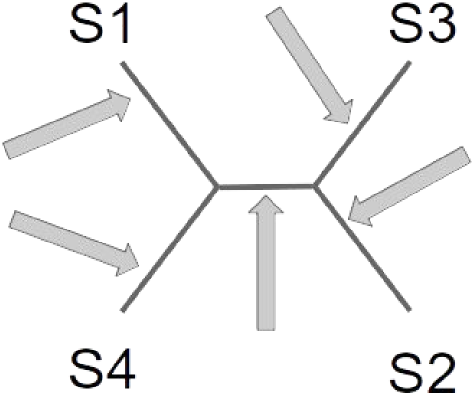
9 changes



7 changes

右边的树以7次变化为最佳，因为它比其他两个选项所需的变化少。因此，在这种特定情况下，将S1和S4放在一起，而将S2作为一个分支添加到S4的路径上，

Heuristic: sequential addition



- prioritise the best four-tip tree
- add S5 in five places
- score each of those trees
- shortcuts: don't recompute unchanged nodes
- continue until we find the best tree
- ***greedy*** approach

Evolutionary Trees II

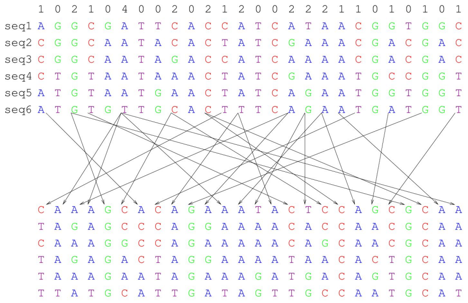
Recap

Distance vs Character methods

Evaluating trees (character methods)
- Fitch algorithm
- Sankoff algorithm

Building trees (character methods)
- Heuristics
- <u>Reliability</u>

Bootstrapping



```
     1 0 2 1 0 4 0 0 2 0 2 1 2 0 0 2 2 2 1 1 0 1 0 1 0 1
seq1 A G G C G A T T C A C C A T C A T A A C G G T G G C
seq2 C G G C A A T A C A C T A T C G A A A C G A C G A C
seq3 C G G C A A T A G A C C A T C A A A A C G A C G A C
seq4 C T G T A A T A A A C T A T C G A A A T G C C G G T
seq5 A T G T A A T G A A C T A T C A G A A T G G T G G T
seq6 A T G T G T T G C A C T T T C A G A A T G A T G G T

     C A A A G C A C A G A A A T A C T C C A G C G C A A
     T A G A G C C C A G G A A A A C A C C A A C G C A A
     C A A A G G C C A G A A A A A C A G C A A C G C A A
     T A G A G A C T A G G A A A A T A A C A C T G C A A
     T A A A G A A T A G A A A G A T G A C A G T G C A A
     T T A T G C A T T G A T A G T T G C C A A T G C A T
```

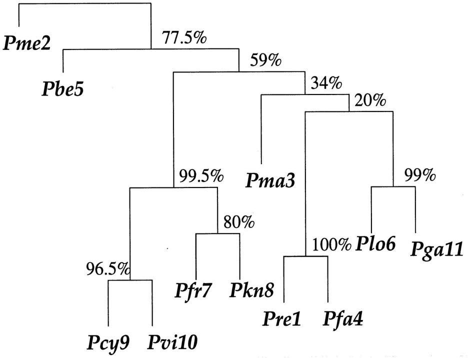
- **general approach**: assess accuracy of an estimator using simulated data
- re-sample columns in an alignment of sequences to create new alignments
- re-apply the same phylogeny reconstruction method

一般方法（General approach）：使用模拟数据评估估计量的准确性。在这里，估计量通常是指构建出的系统发育树。

重新采样序列中的列（Re-sample columns in an alignment of sequences）：在一组序列比对中随机选取列（也就是序列的位置），并允许重复选择同一列，从而创建新的序列比对。

重新应用相同的系统发育重建方法（Re-apply the same phylogeny reconstruction method）：使用新的序列比对，重复系统发育树的构建过程。

Bootstrapping



- 🌲 repeat bootstrapping (at least 100 times)
- 🌲 count occurrence of nodes in bootstrap trees
- 🌲 if we see the branching point often, it is more reliable
- 🌲 ***rule of thumb***: accept bootstrap values from 90–100 %

Thank you!

重复自助法（至少100次）（repeat bootstrapping (at least 100 times)）：这意味着进行至少100次自助重采样，以生成一定数量的系统发育树，以此来评估各个分支点的可信度。

计算节点在自助树中出现的次数（count occurrence of nodes in bootstrap trees）：通过统计每个节点（即分支点）在所有生成的自助树中出现的次数，可以得到每个节点的支持度。

如果经常看到分支点，它更可靠（if we see the branching point often, it is more reliable）：这说明如果一个分支点在多次自助法分析中经常出现，那么我们可以认为这个分支点是可靠的。

经验法则：接受自助值从90-100%（rule of thumb: accept bootstrap values from 90 – 100%）：这是一个通用的准则，建议接受那些支持值在90%到100%之间的分支点，因为这些值表示高可信度。

Today: Evolutionary Trees II

Next time: Genomic Features & Regions