Parsimony

Large Parsimony, Search Algorithms, Branch confidence

Genome 559: Introduction to Statistical and Computational Genomics

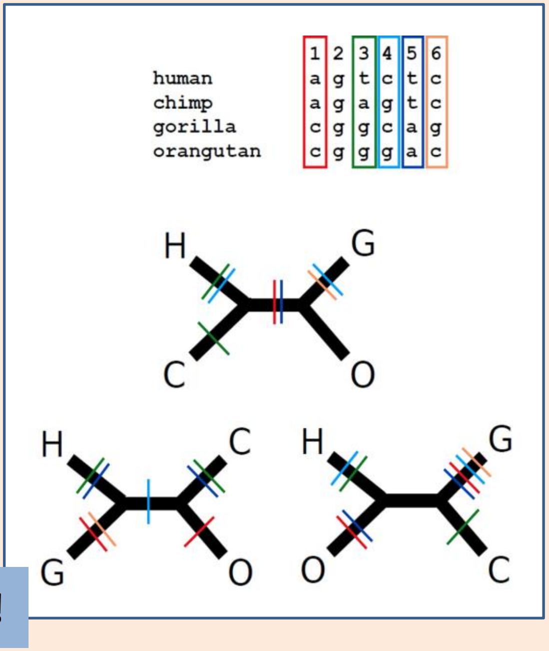
Elhanan Borenstein

A quick review

- The parsimony principle:
  - Find the tree that requires the fewest evolutionary changes!
- A fundamentally different method:
  - Search rather than reconstruct
- Parsimony algorithm
  1. Construct all possible trees
  2. For each site in the alignment and for each tree count the minimal number of changes required
  3. Add sites to obtain the total number of changes required for each tree
  4. Pick the tree with the lowest score

Too many!

The small parsimony problem

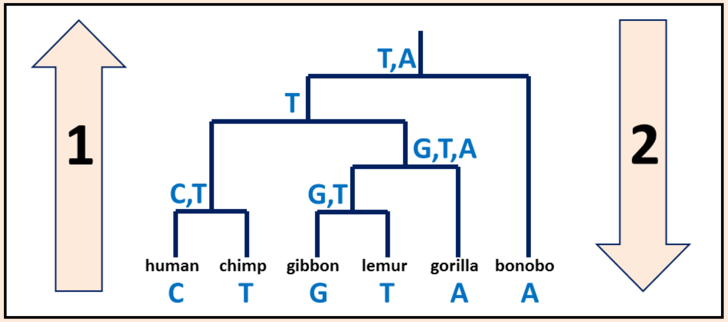


	1	2	3	4	5	6
human	a	g	t	c	t	c
chimp	a	g	a	g	t	c
gorilla	c	g	g	c	a	g
orangutan	c	g	g	g	a	c

A quick review – cont’

- Small vs. large parsimony
  - **Large parsimony:** Find the topology which gives best score
  - **Small parsimony**: Given a tree topology and the state in all the tips, find the minimal number of changes required

- Fitch’s algorithm:
  1. **Bottom-up phase**: Determine the set of possible states
  2. **Top-down phase**: Pick a state for each internal node



And now back to the “big” parsimony problem …

How do we find the most parsimonious tree amongst the ***many*** *possible trees?*

Searching tree space

- **Exhaustive search**:

  Up to 8-10 leaves (10k-2m unrooted trees, 135k-34m rooted)

  Guaranteed results

- **Branch-and-bound*:**

  Up to 10-20 leaves

  Guaranteed results!!!

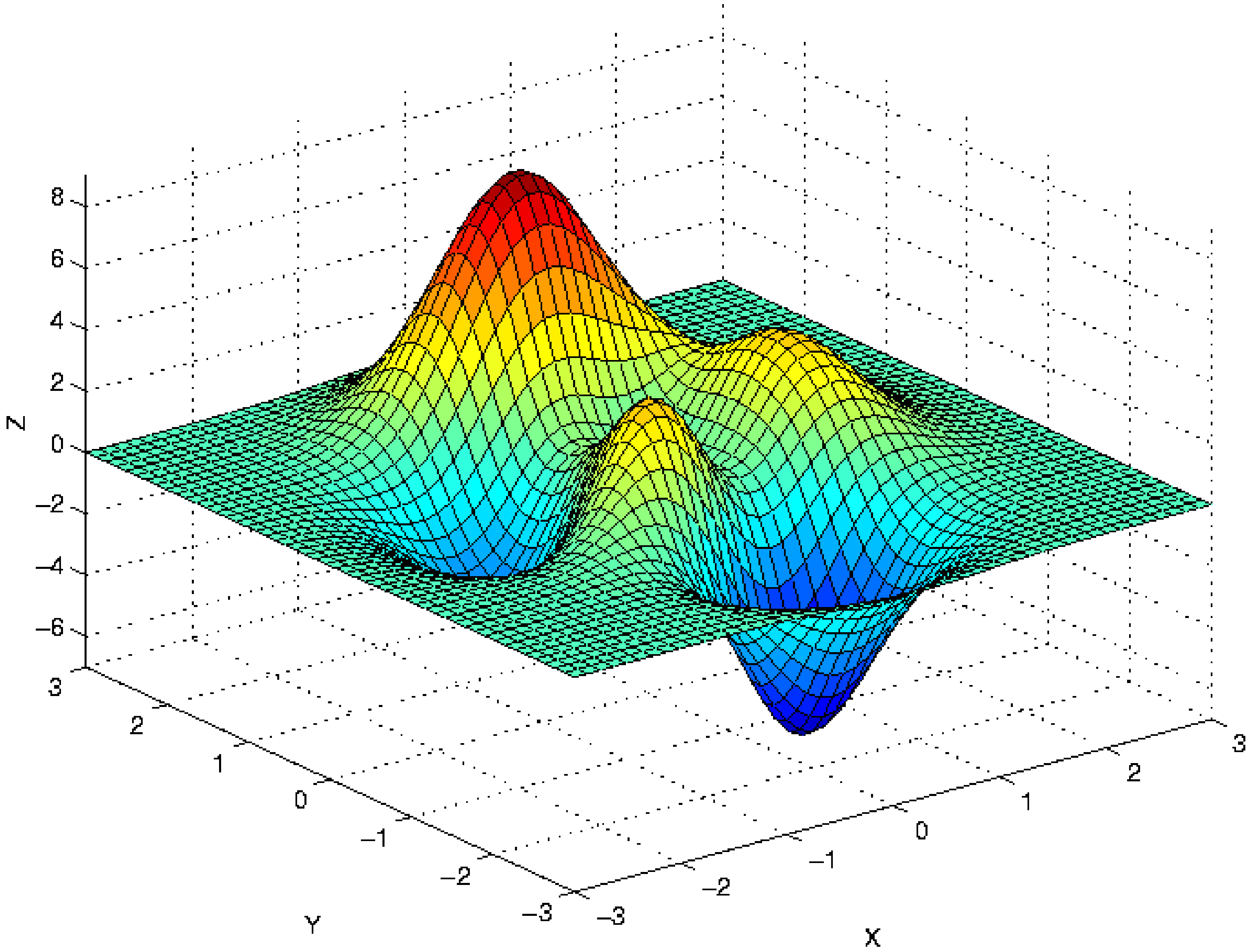
  * Branch-and-bound is a clever way of ruling out most trees as they are built, so you can evaluate more trees by exhaustive search.

- **Heuristic search** (e.g. hill-climb)**:**

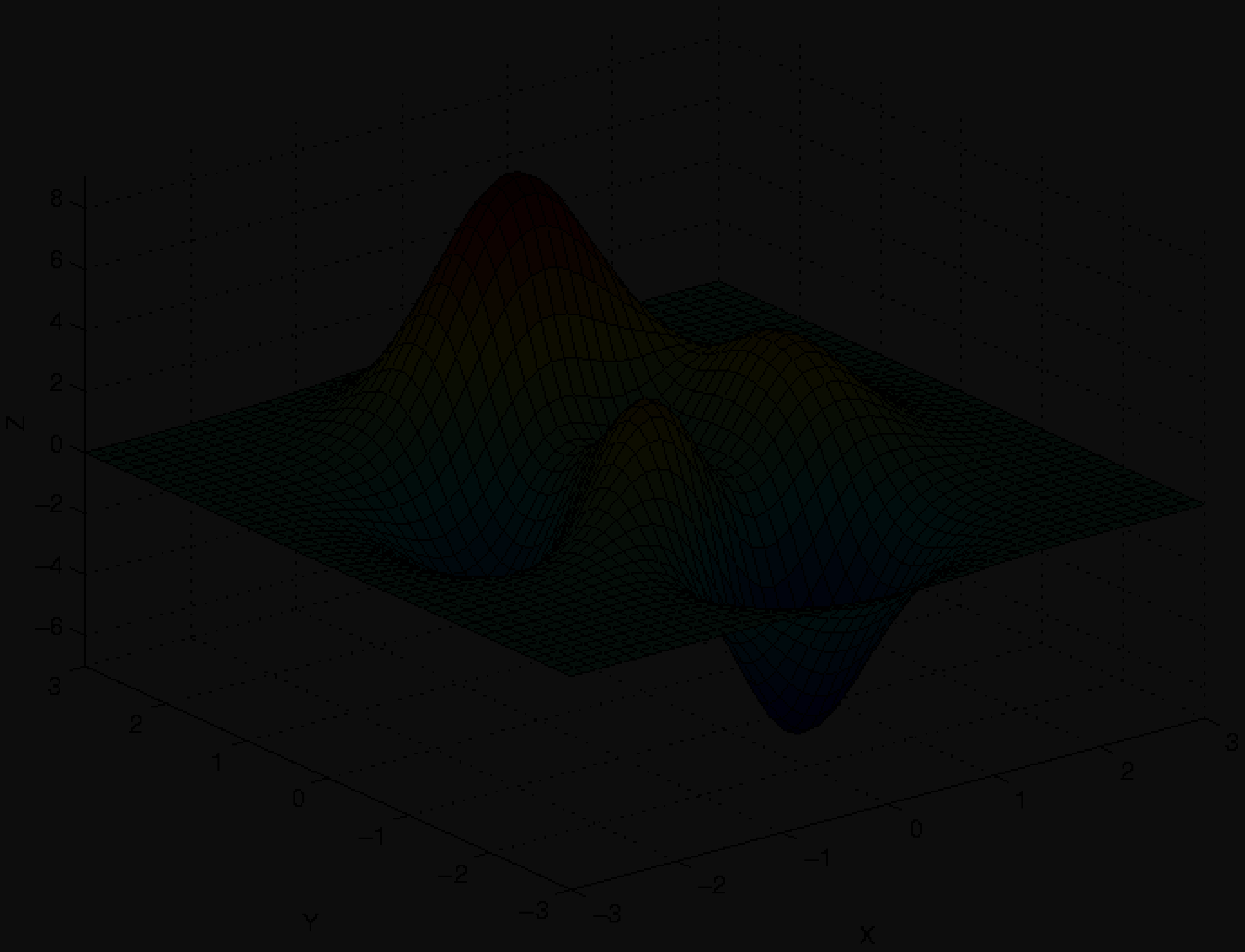
  20+ leaves

  May not find correct solution.

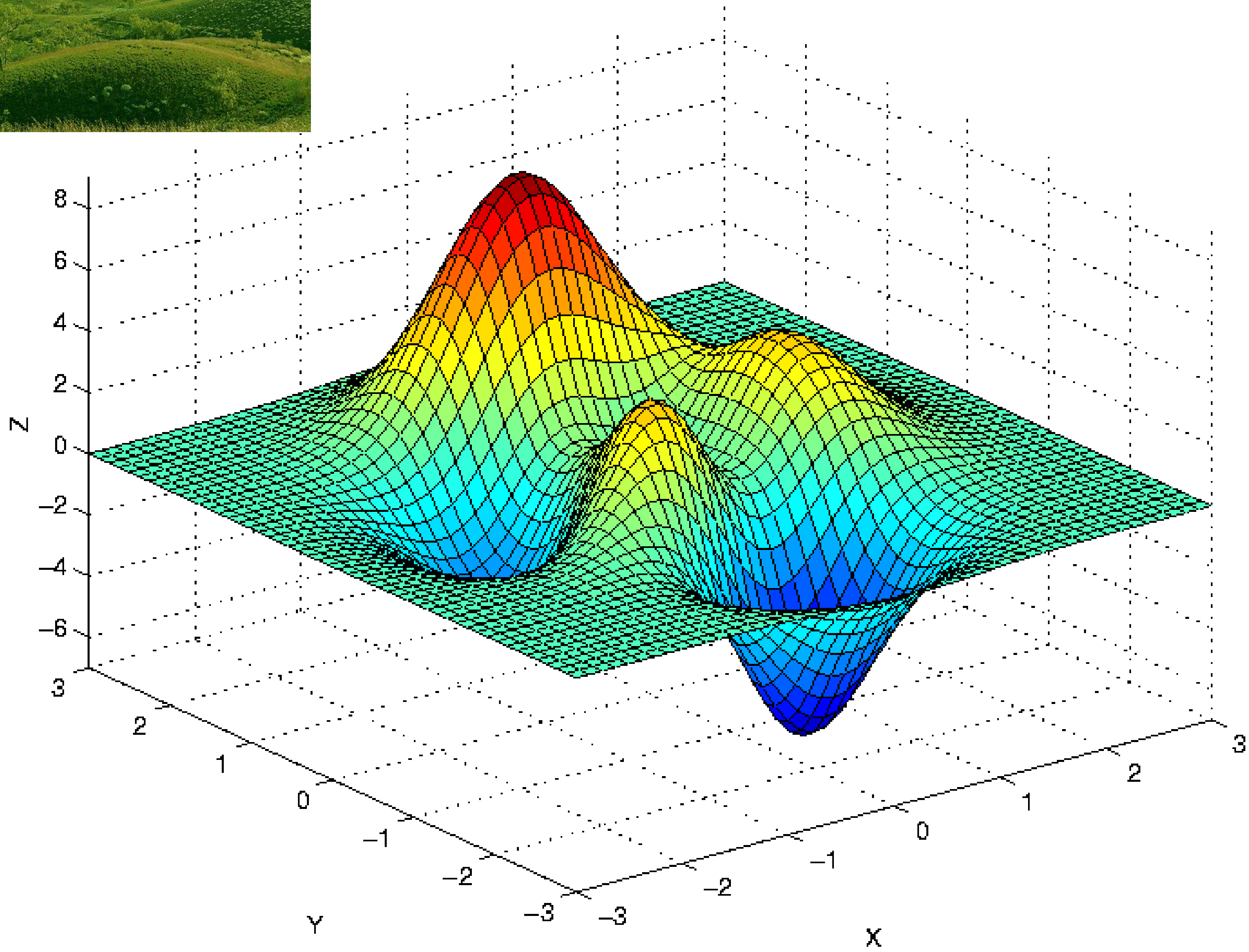
Search space



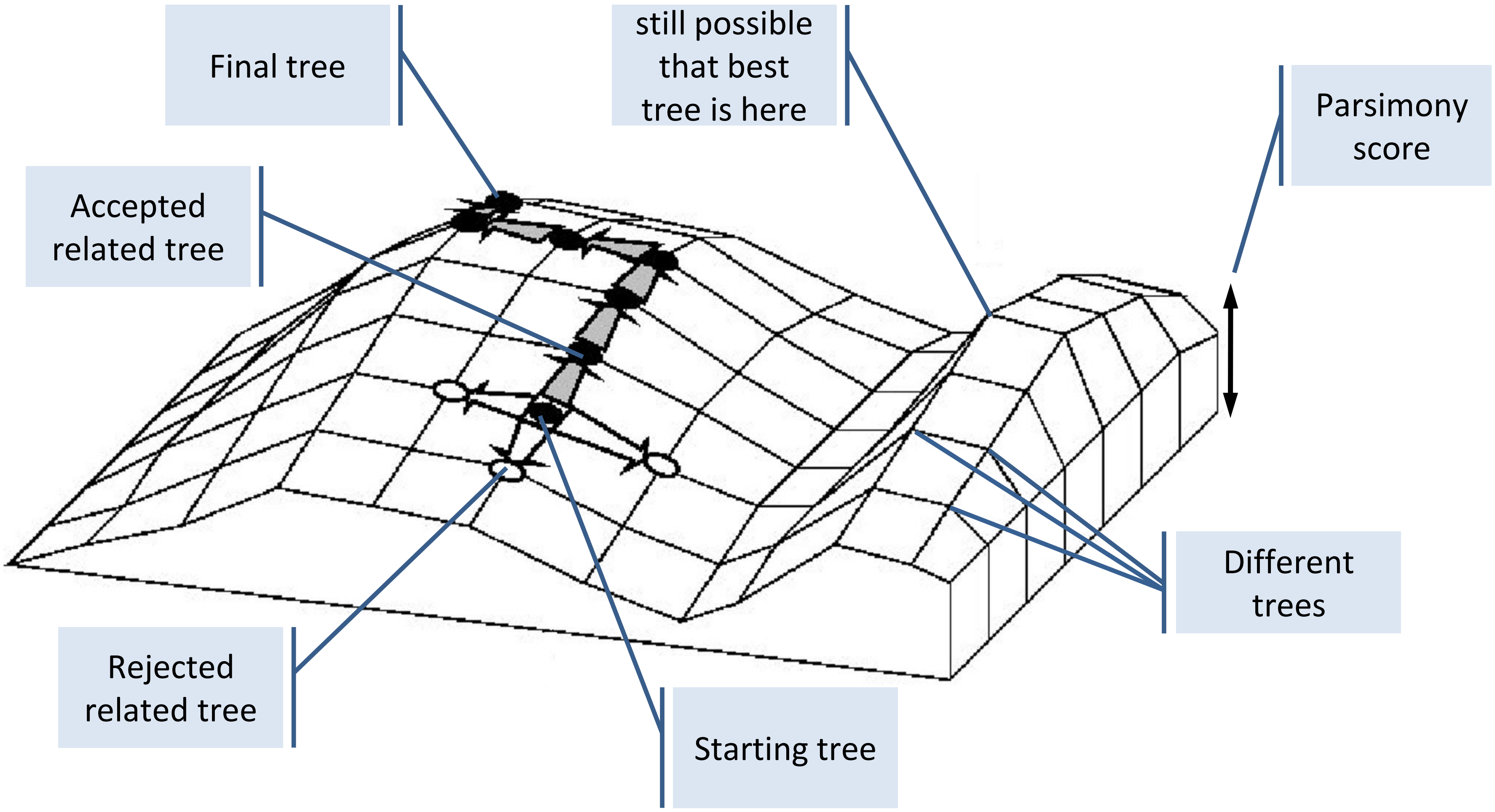
Search space



Hill-climbing



Hill-climbing for searching “best” tree



Final tree

still possible that best tree is here

Parsimony score

Accepted related tree

Different trees

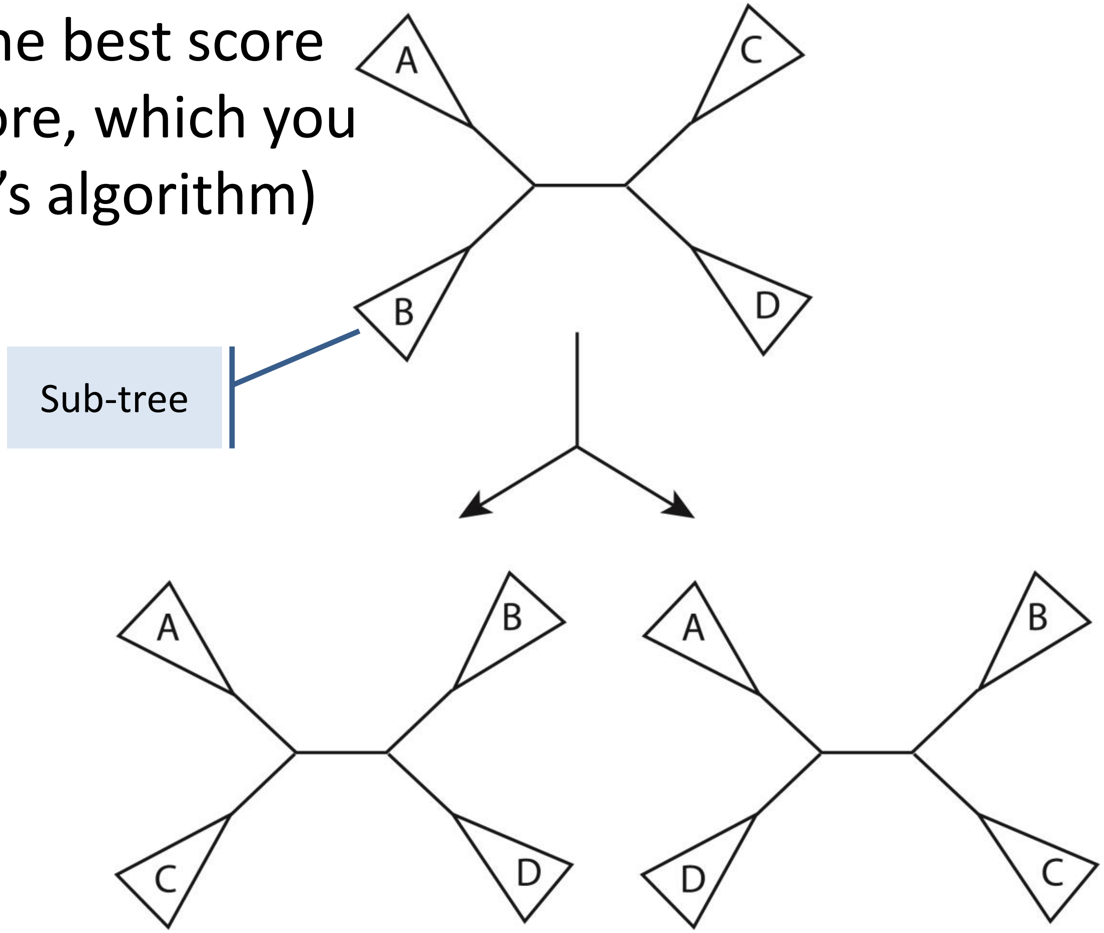
Rejected related tree

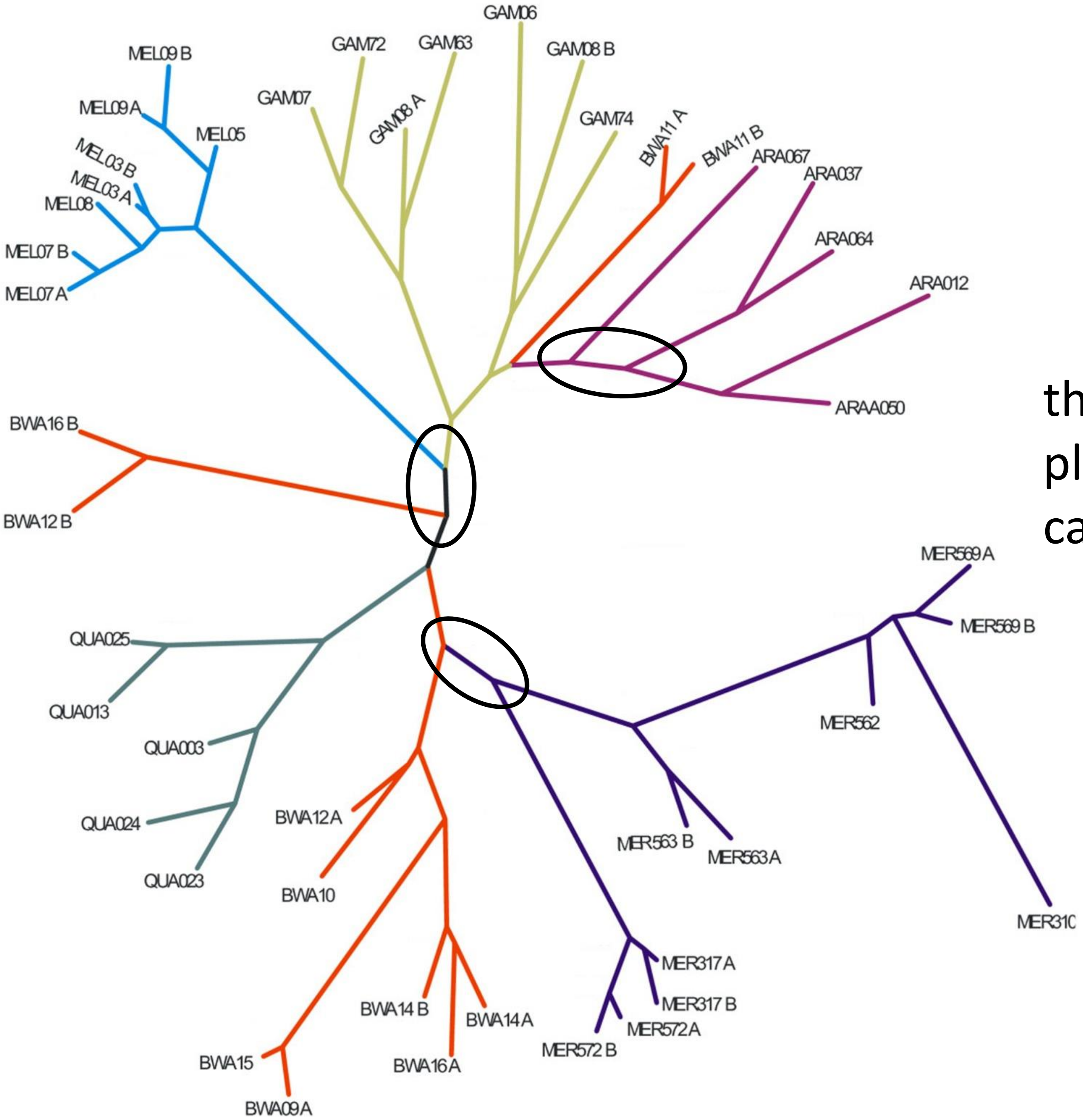
Starting tree

A “greedy” algorithm

Nearest-Neighbor Interchange (NNI)

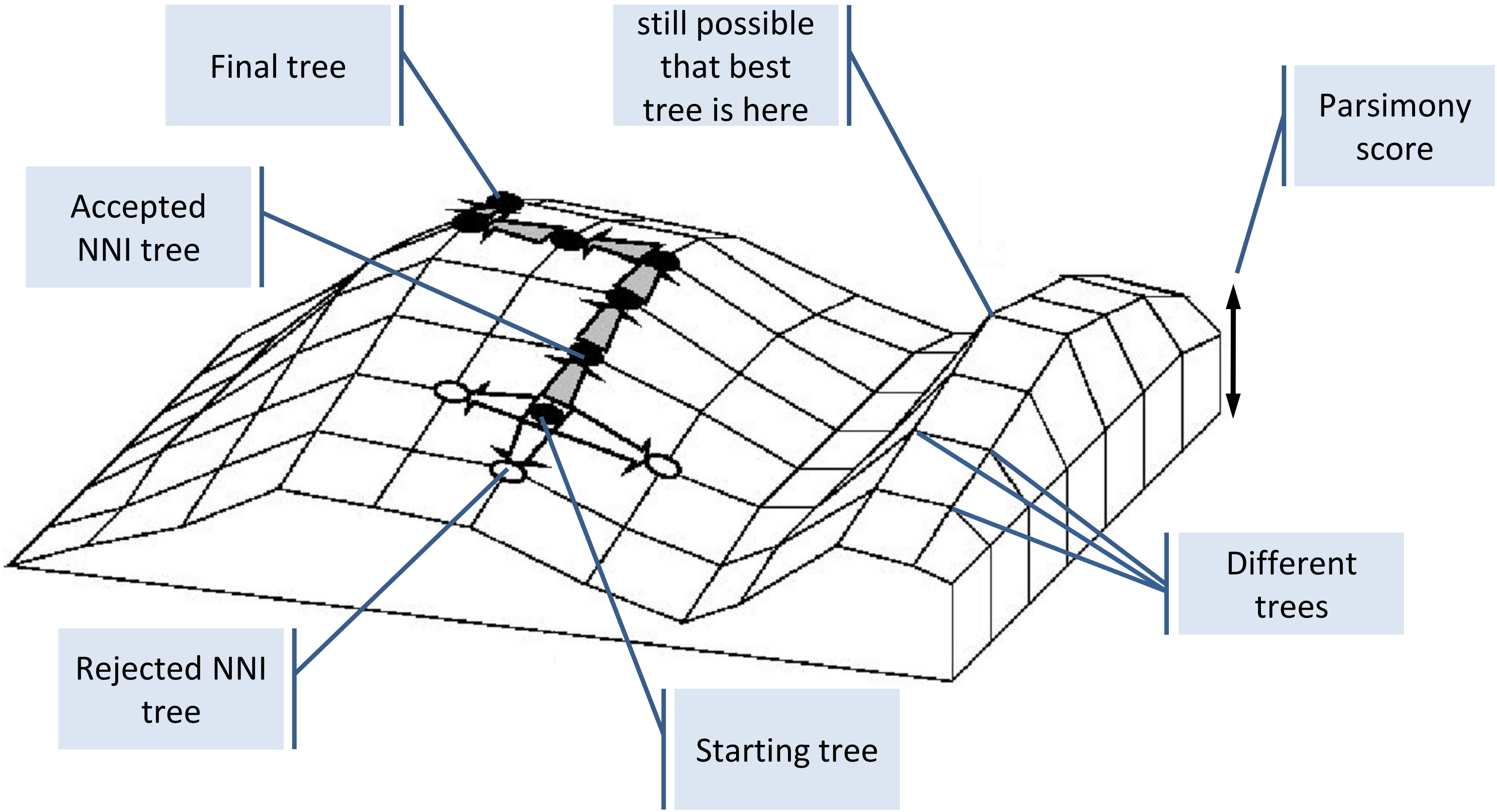
1. Find a tree with some score.
2. At **each internal branch** consider the two alternative arrangements of the 4 sub-trees.
3. Keep the tree that has the best score (e.g., best parsimony score, which you can calculate using Fitch’s algorithm)
4. Repeat.





three (of many) places where NNI can be considered

Hill-climbing with NNI



Final tree

still possible that best tree is here

Parsimony score

Accepted NNI tree

Different trees

Rejected NNI tree

Starting tree

A “greedy” algorithm

The parsimony algorithm

1) *Construct all possible trees* ***or search the space of possible trees using NNI hill-climb***

2) *For each site in the alignment and for each tree count the minimal number of changes required* ***using Fitch's algorithm***

3) *Add all sites up to obtain the total number of changes for each tree*

4) *Pick the tree with the lowest score* ***or search until no better tree can be found***

How can we improve this algorithm and increase our chances of finding the optimal tree?

Phylogenetic trees: Summary

Parsimony Trees:

1) *Construct all possible trees* ***or search the space of possible trees***
2) *For each site in the alignment and for each tree count the minimal number of changes required* ***using Fitch's algorithm***
3) *Add all sites up to obtain the total number of changes for each tree*
4) *Pick the tree with the lowest score*

Distance Trees:

1) *Compute pairwise corrected distances.*
2) *Build tree by sequential clustering algorithm (UPGMA or Neighbor-Joining).*
3) *These algorithms don't consider all tree topologies, so they are very fast, even for large trees.*

Maximum-Likelihood Trees:

1) *Tree evaluated for likelihood of data given tree.*
2) *Uses a specific model for evolutionary rates (such as Jukes-Cantor).*
3) *Like parsimony, must search tree space.*
4) *Usually most accurate method but slow.*

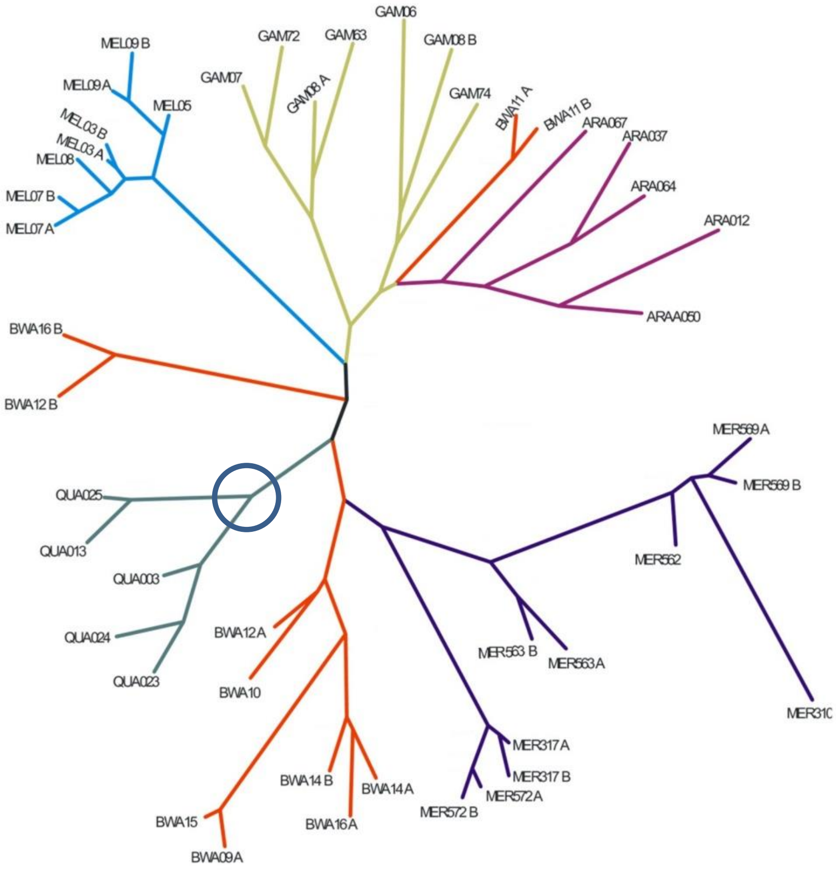
Branch confidence

How certain are we that this is the correct tree?

Can be reduced to many simpler questions - how certain are we that each **branch point** is correct?

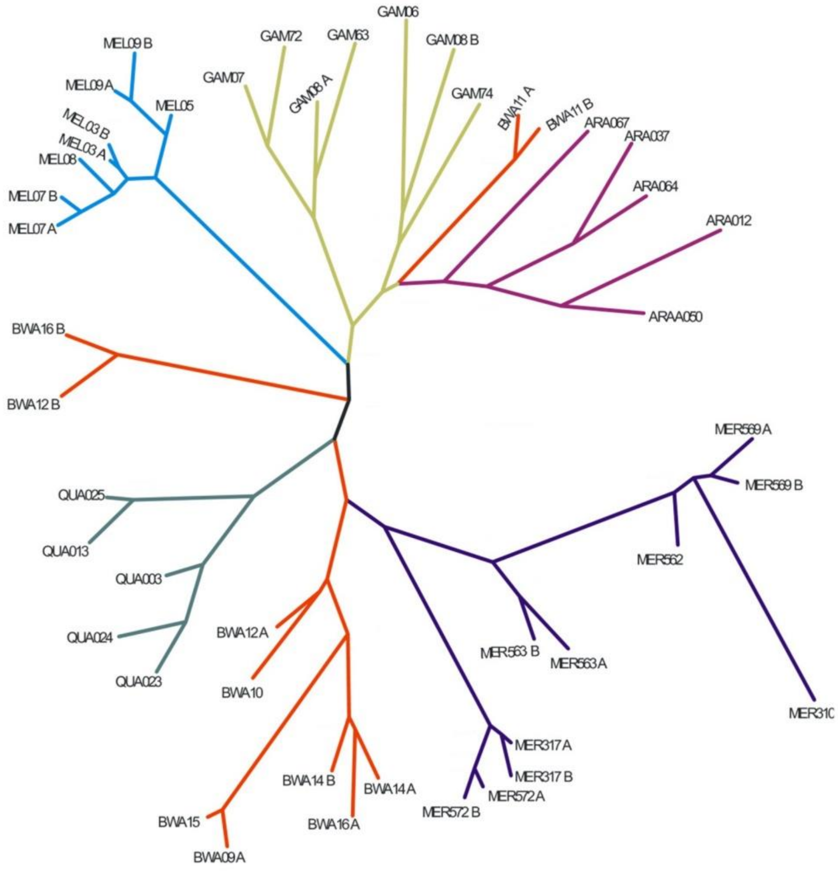
For example, at the circled branch point, how certain are we that the three subtrees have the correct content:

subtree1 - QUA025, QUA013
subtree2 - QUA003, QUA024, QUA023
subtree3 - everything else



Branch confidence

What if I had multiple datasets (e.g., multiple alignments)?

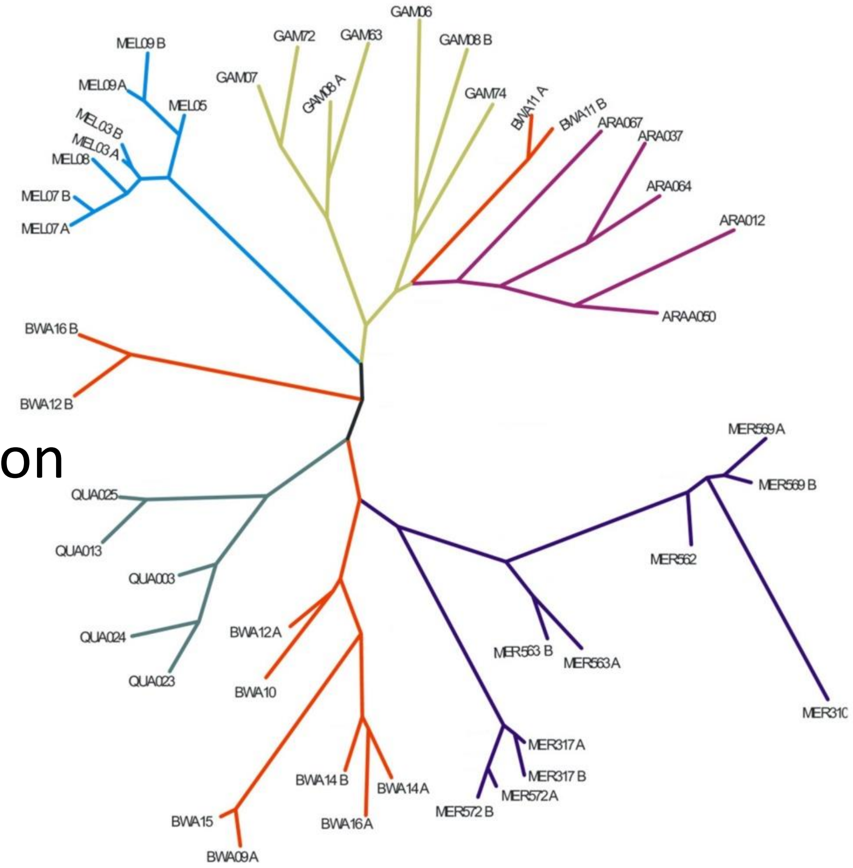


Branch confidence

What if I had multiple datasets (e.g., multiple alignments)?

1. Infer a tree from each dataset

2. For each branch point on the computed tree, count what fraction of trees have the same subtree partitions (regardless of topology within the subtrees).



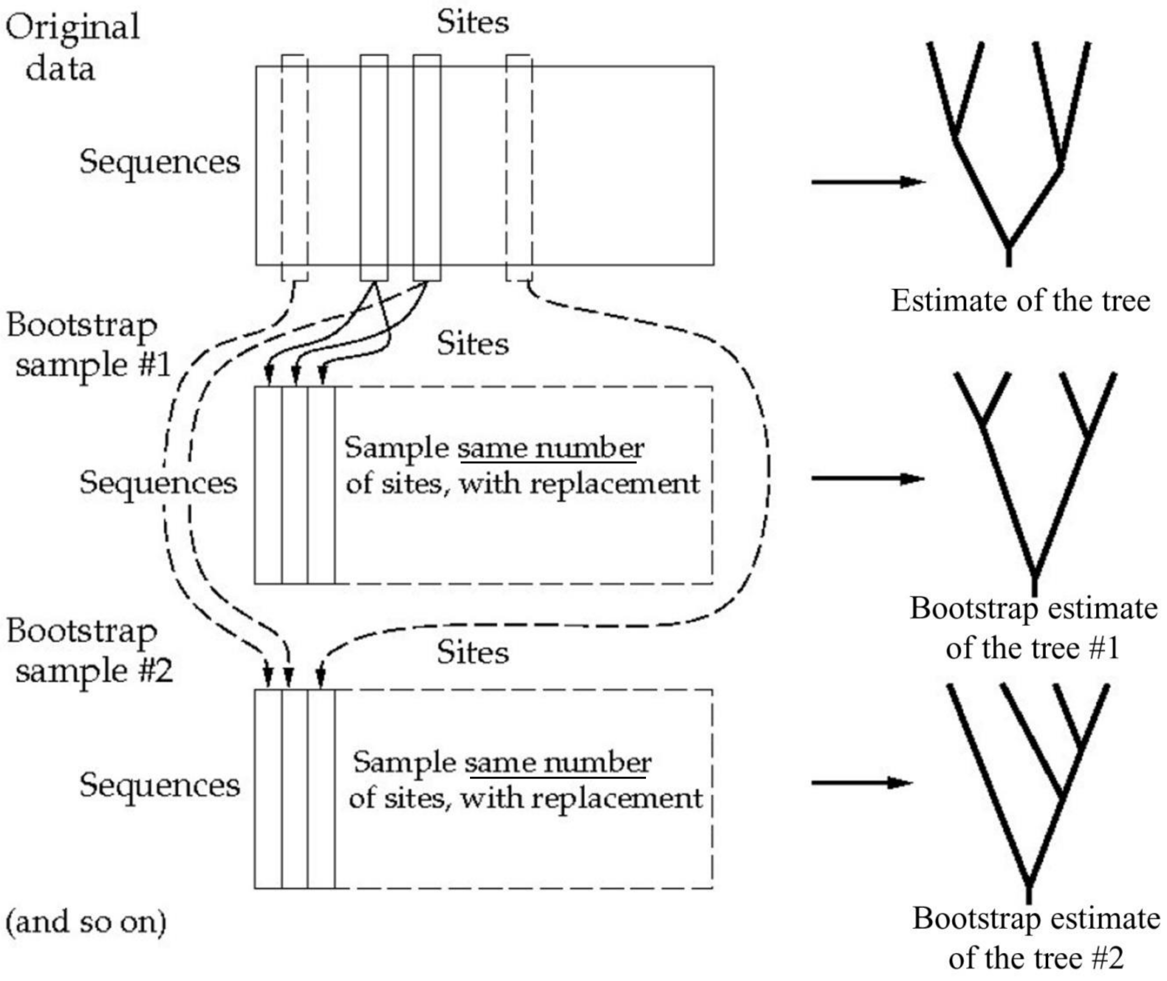
Bootstrap support

Most commonly used branch support test:

1. Randomly sample alignment sites.

2. Use sample to estimate the tree.

3. Repeat many times.


Original data
Sites
Sequences
Estimate of the tree
Bootstrap sample #1
Sites
Sequences
Sample same number of sites, with replacement
Bootstrap estimate of the tree #1
Bootstrap sample #2
Sites
Sequences
Sample same number of sites, with replacement
Bootstrap estimate of the tree #2
(and so on)

(sample with replacement means that a sampled site remains in the source data after each sampling, so that some sites will be sampled more than once)

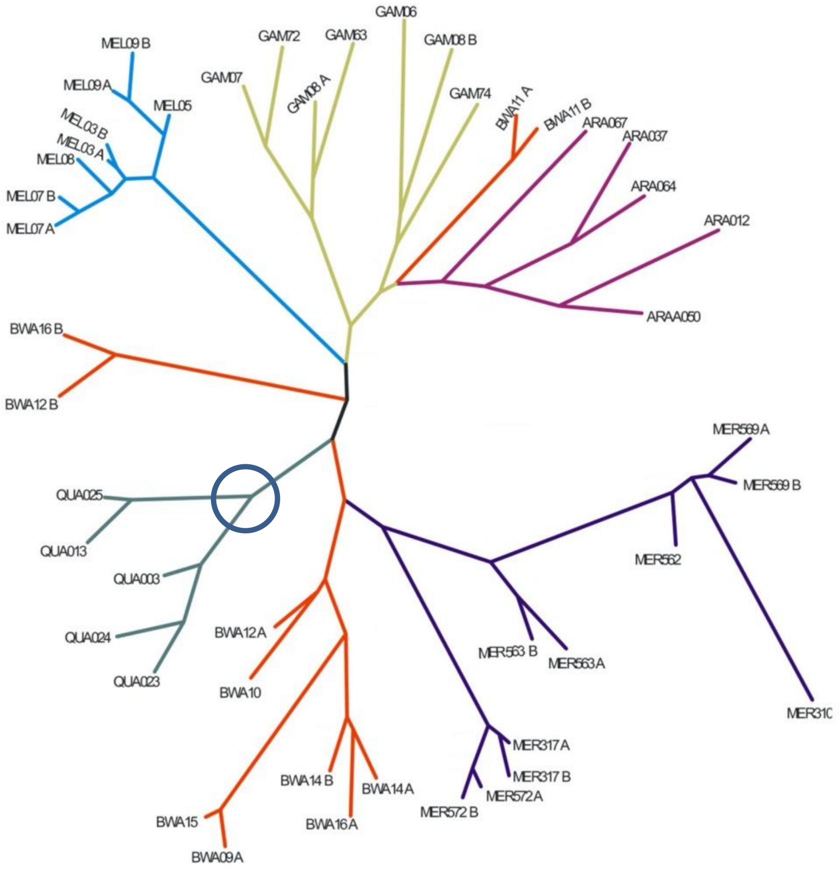
Bootstrap support

For each branch point on the computed tree, count what fraction of the bootstrap trees have the same subtree partitions (regardless of topology within the subtrees).

For example at the circled branch point, what fraction of the bootstrap trees have a branch point where the three subtrees include:

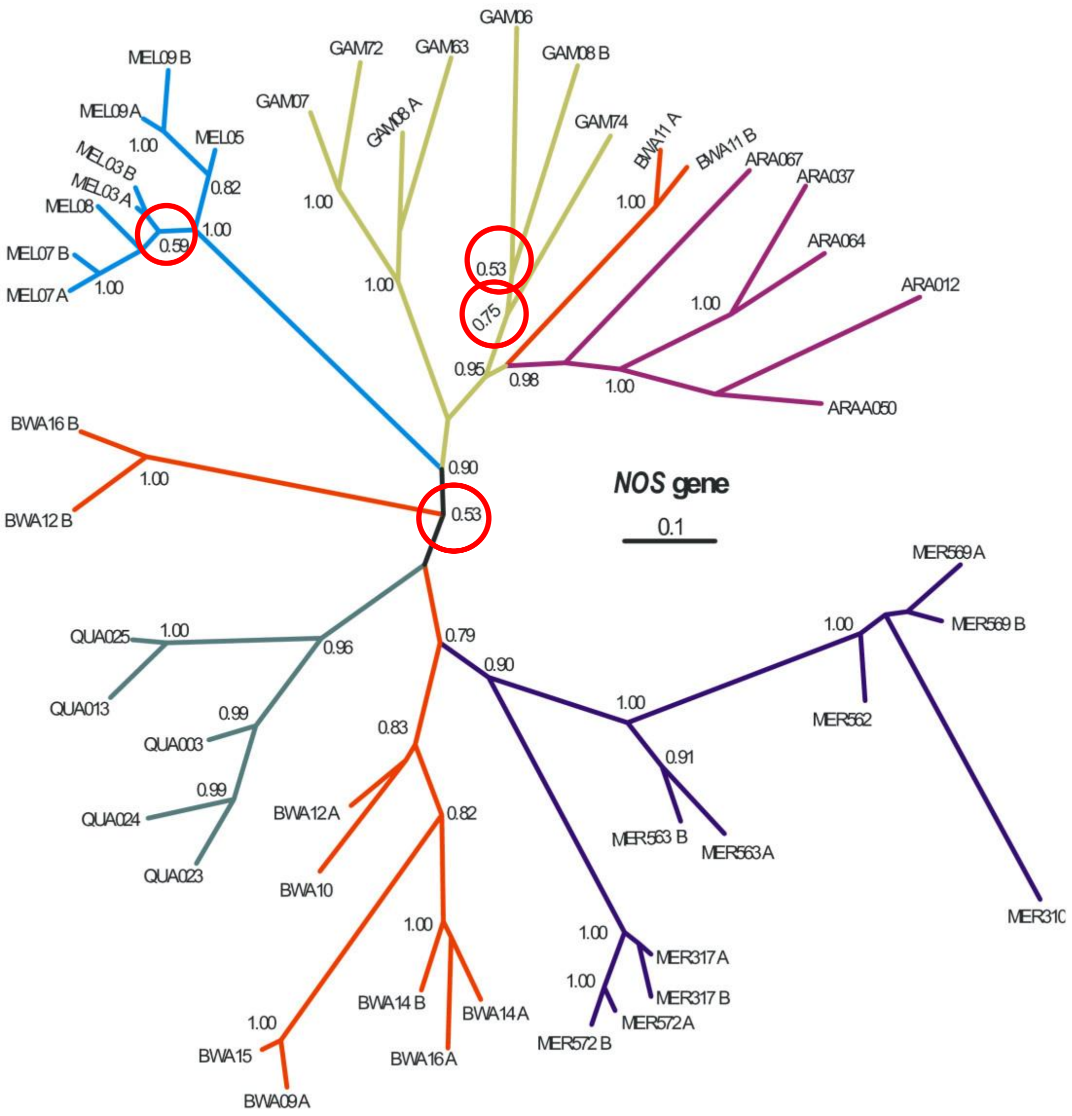
subtree1 - QUA025, QUA013
subtree2 - QUA003, QUA024, QUA023
subtree3 - everything else

This fraction is the **bootstrap support** for that branch.



Original tree figure with branch supports

(here as fractions, also common to give % support)



low-confidence branches are marked