Parsimony II
Search Algorithms

Genome 373
Genomic Informatics
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
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 — Too many!
  2. For each site in the alignment and for each tree count the minimal number of changes required — The small parsimony problem
  3. Add sites to obtain the total number of changes required for each tree
  4. Pick the tree with the lowest score
A quick review – cont’

- Small vs. large parsimony

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

\[
R_i = \begin{cases} 
    \text{if } R_j \cap R_k \neq \emptyset \rightarrow R_j \cup R_k \\
    \text{otherwise} \rightarrow R_j \cup R_k 
\end{cases}
\]

\[
s_i = \begin{cases} 
    \text{if } s_j \in R_i \rightarrow s_j \\
    \text{otherwise} \rightarrow \text{arbitrary state } \in R_i 
\end{cases}
\]

Parsimony-score = # union operations

Parsimony-score = 4
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.
Hill-climbing
Hill-climbing

A “greedy” algorithm

Final tree

Accepted related tree

Rejected related tree

Starting tree

still possible that best tree is here

Different trees

Parsimony score
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.
4. Repeat.
three (of many) places where NNI can be considered
Hill-climbing with NNI

Accepted NNI tree

Rejected NNI tree

Starting tree

Final tree

Different trees

Parsimony score

still possible that best tree is here

A “greedy” algorithm
How can we improve this algorithm and increase our chances of finding the optimal tree?
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
## Phylogenetic trees: Summary

<table>
<thead>
<tr>
<th>Parsimony Trees:</th>
<th>Distance Trees:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Construct all possible trees or search the space of possible trees</td>
<td>1) Compute pairwise corrected distances.</td>
</tr>
<tr>
<td>2) For each site in the alignment and for each tree count the minimal number of changes required using</td>
<td>2) Build tree by sequential clustering algorithm (UPGMA or Neighbor-Joining).</td>
</tr>
<tr>
<td>Fitch's algorithm</td>
<td>3) These algorithms don't consider all tree topologies, so they are very fast, even for large trees.</td>
</tr>
<tr>
<td>3) Add all sites up to obtain the total number of changes for each tree</td>
<td></td>
</tr>
<tr>
<td>4) Pick the tree with the lowest score</td>
<td></td>
</tr>
</tbody>
</table>

## 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
Bootstrap support

Most commonly used branch support test:

1. Randomly sample alignment sites (with replacement).
2. Use sample to estimate the tree.
3. Repeat many times.

(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