Some of these slides have been borrowed from Dr. Paul Lewis, Dr. Joe Felsenstein. Thanks!

Paul has many great tools for teaching phylogenetics at his web site:

http://hydrodictyon.eeb.uconn.edu/people/plewis
Tree Searching

Parsimony and ML give us ways to deciding whether one tree is fits our data better than another tree, but . . .

How do we find the best tree?
(or one that is good enough)
Exhaustive Enumeration

With the first three taxa, create the trivial unrooted tree.
Exhaustive Enumeration...

Can add fourth taxon (D) to any of the three edges

© 2007 by Paul O. Lewis
Exhaustive Enumeration
(getting tired yet?)

Can add fifth taxon (E) to any of the 5 edges of each of the 3 4-taxon trees!
<table>
<thead>
<tr>
<th>Tips</th>
<th>Number of unrooted (binary) trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>105</td>
</tr>
<tr>
<td>7</td>
<td>945</td>
</tr>
<tr>
<td>8</td>
<td>10,395</td>
</tr>
<tr>
<td>9</td>
<td>135,135</td>
</tr>
<tr>
<td>10</td>
<td>2,027,025</td>
</tr>
<tr>
<td>11</td>
<td>34,459,425</td>
</tr>
<tr>
<td>12</td>
<td>654,729,075</td>
</tr>
<tr>
<td>13</td>
<td>13,749,310,575</td>
</tr>
<tr>
<td>14</td>
<td>316,234,143,225</td>
</tr>
<tr>
<td>15</td>
<td>7,905,853,580,625</td>
</tr>
<tr>
<td>16</td>
<td>213,458,046,676,875</td>
</tr>
<tr>
<td>17</td>
<td>6,190,283,353,629,375</td>
</tr>
<tr>
<td>18</td>
<td>191,898,783,962,510,625</td>
</tr>
<tr>
<td>19</td>
<td>6,332,659,870,762,850,625</td>
</tr>
<tr>
<td>20</td>
<td>22,164,309,5476,699,771,875</td>
</tr>
<tr>
<td>21</td>
<td>8,200,794,532,637,891,559,375</td>
</tr>
<tr>
<td>22</td>
<td>319,830,986,772,877,770,815,625</td>
</tr>
<tr>
<td>23</td>
<td>13,113,070,457,687,988,603,440,625</td>
</tr>
<tr>
<td>24</td>
<td>563,862,029,680,583,509,947,946,875</td>
</tr>
</tbody>
</table>
For $N$ taxa:

\[
\# \text{ unrooted, binary trees} = \prod_{i=3}^{N-1} (2i - 3)
\]

\[
= \prod_{i=4}^{N} (2i - 5)
\]

\[
\# \text{ rooted, binary trees} = \prod_{i=3}^{N} (2i - 3)
\]

\[
= (2N - 3)(\# \text{ unrooted, binary trees})
\]
Star decomposition
Star decomposition
Star decomposition
Star decomposition

- Very greedy. Once a pair of species are joined, they will not be separated.

- Neighbor-joining is star decomposition under the balanced minimum evolution criterion
Number of trees scored in the first step of star decomposition

- + # trees in first step

# leaves
# trees scored
Log–Log Plot of the number of trees scored in star decomposition

* Log total # trees
+ Log # trees in first step
Star decomposition

Number of trees scored for $N$ taxa (if we decrease $i$ in the summation):

$$\#	ext{ trees scored} = 3 \sum_{i=N}^{5} \binom{i}{2}$$

$$= 3 \sum_{i=N}^{5} \frac{i(i - 1)}{2}$$

Thus, star decomposition is $O(N^3)$.

For $N=10$:

$$158 = 45 + 36 + 28 + 21 + 15 + 10 + 3$$
Star decomposition can fail even when there is no homoplasy

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>000000</td>
</tr>
<tr>
<td>B</td>
<td>000000</td>
</tr>
<tr>
<td>C</td>
<td>110000</td>
</tr>
<tr>
<td>D</td>
<td>110000</td>
</tr>
<tr>
<td>E</td>
<td>101011</td>
</tr>
<tr>
<td>F</td>
<td>101111</td>
</tr>
<tr>
<td>G</td>
<td>1011??</td>
</tr>
</tbody>
</table>

Star decomposition (under parsimony) will join E and F in the first step based on characters 5 and 6 (shown in red), even though (((A,B),(C,D)),E,(F,G)) shows no homoplasy.
Stepwise addition
Stepwise addition
Stepwise addition
Stepwise addition

- Greedy, but can introduce a new taxon on the path between taxa that have already been joined.

- The tree can depend on the input order of the taxa

- Number of trees scored for $N$ taxa:

  $$\# \text{ trees scored} = \sum_{i=3}^{N-1} (2i - 3) = (N - 1)(N - 3)$$

  Thus, stepwise addition is $O(N^2)$. For $N=10$:

  $$63 = 3 + 5 + 7 + 9 + 11 + 13 + 15$$
Construct and score an initial estimate of the tree
Branch and bound

Start building a new tree of three taxa
Branch and bound

Consider all positions for the next taxon

Initial tree
Branch and bound

Score all of these trees

Initial tree
Branch and bound

Consider adding the next taxon to all viable backbone trees.
Branch and bound

- Guaranteed to find all of the optimal trees.

- Relies on the fact that the score always gets worse as you add taxa.

- Can be as fast as stepwise addition (if you have lots of very clean data).

- Can be as slow as an exhaustive search (if you have little data and/or data with lots of conflict).

- Bound can be tightened by adding a lower bound on the number of steps to be added when you add more taxa to the tree (based on new forms of character conflict in unattached taxa).
Trying to improve a tree

Neither stepwise addition is guaranteed to return the best tree(s), but branch-and-bound (or exhaustive searching) is frequently infeasible.

Heuristic hill-climbing searches can work quite well:

1. Start with a tree
2. Score the tree
3. Consider a new tree within the neighborhood of the current tree:
   (a) Score the new tree.
   (b) If the new tree has a better tree, use it as the “current tree”
   (c) Stop if there are no other trees within the neighborhood to consider.

These are not guaranteed to find even one of the optimal trees.

The most common way to explore the neighborhood of a tree is to swap the branches of the tree to construct similar trees.
Greedy search for a maximum

If start here
Greedy search for a maximum

If start here
Greedy search for a maximum

If start here
Greedy search for a maximum

If start here
Greedy search for a maximum

If start here
Greedy search for a maximum

If start here
Greedy search for a maximum

If start here
Greedy search for a maximum

If start here
Greedy search for a maximum

If start here
Greedy search for a maximum

If start here
Greedy search for a maximum

If start here
Greedy search for a maximum
If start here

Week 2: Searching for trees, ancestral states – p.13/51
Greedy search for a maximum

If start here
Greedy search for a maximum

If start here
Greedy search for a maximum

If start here

Week 2: Searching for trees, ancestral states – p.16/51
Greedy search for a maximum

end up here

If start here
Greedy search for a maximum

end up here  but global maximum is here

If start here
Nearest-neighbor rearrangements

is rearranged by dissolving the connections to an interior branch

and reforming them in one of the two possible alternative ways:
Nearest Neighbor Interchanges searches

1. Consider the two possible NNI neighbors “around” each internal edge in the tree
2. Return the tree set of trees that is at least as good as all of the NNI neighbors.
3. Number of rearrangements scored per tree \( N \) taxa:

\[
\# \text{ rearrangements scored} = 2 \times (N - 3)
\]

But, there is no upper bound on the number of trees encountered on the path from the initial tree to the final tree.
Schoenberg graph – edges connect NNI neighbors
Tree “Islands” possible

An $Op - L$ tree island (sensu ?): A set of trees with score $\leq L$ that are connected to each other by $Op$ operations such that you can get from any tree in the set to any other tree by repeated $Op$ changes and all intermediate trees along the path are also members of the set.

The following Schoenberg graph shows the scores of the 15 trees on the following dataset (contrived data by POL):

A  ACGCAGGT
B  ATGGTGAT
C  GCTCACGG
D  ACTGTCGT
E  GTTCTGAG
Schoenberg graph with parsimony scores
Schoenberg graph showing the single NNI-15 island
Schoenberg graph showing the single NNI-14 island
Schoenberg graph showing the both NNI-13 islands
Tree Islands implications

1. Islands can be larger than 1 tree – so we must consider ties if we want to find all trees that optimize the score.
2. There can be more than 1 island for good scores – so even swapping to completion on all optimal trees found in a search is guaranteed to succeed.
3. The delimitation of an island depends on tree changing operation used.
Heuristics explore “Tree Space”

Most commonly used methods are “hill-climbers.”

Multiple optima found by repeating searches from different origins.

Severity of the problem of multiple optima depends on step size.
Subtree Pruning Regrafting (SPR) and Tree Bisection Reconnection (TBR)

SPR maintains subtree rooting

TBR tries all possible rootings
1-Edge-contract Refine
2-Edge-contract Refine
Many other heuristic strategies proposed

- Swapping need not include *all* neighbors (RAxML, reconlimit in PAUP*)
- “lazy” scoring of swaps (RAxML)
- Ignoring (at some stage) interactions between different branch swaps (PHYML)
- Stochastic searches
  - Genetic algorithms (GAML, MetaPIGA, GARLI)
  - Simulated annealing
- Divide and conquer methods (the sectortial searching of Goloboff, 1999; Rec-I-DCM3 Roshan 2004)
- Data perturbation methods (e.g. Kevin Nixon’s “ratchet”)
Population with variation
Population with variation

lnL calculated:
-127.5
-128.1
-131.0
-131.6
-132.0
Population with variation

lnL calculated

Fitness calculated

0.623

0.341

0.019

0.010

0.007
Population with variation

InL calculated

Fitness calculated

Selection
Population with variation

\[ \ln L \text{ calculated} \]

Fitness calculated

Selection

Mutation
Divide-and-Conquer Methods

The basic outline of a phylogenetic Divide-and-Conquer approach is:

1. **Decompose** a starting tree into subsets of the taxa.

2. **Improve** the tree for each of the subsets of taxa.

3. **Merge** the resulting trees into a tree for the full set of taxa.

4. **Refine** the full tree (it will often have polytomies).

5. **Improve** the full tree using a simple (and fast) heuristic.

Examples include Rec-I-DCM3 by Roshan *et al.* (2004). See Goloboff and Pol (*Systematic Biology*, 2007) for a contrasting viewpoint about the relative efficiency of Rec-I-DCM3 compared to heuristics implemented in TNT.
Step 1: Leaf set decomposition

In Rec-I-DCM3 Roshan et al. (2004):

- A tree is divided ("decomposed") into 4 trees around a central edge. The edge is chosen such that it comes as close as possible to dividing the taxa into 2 equally-sized groups.

- The short quartet (taxa closest to this edge in each of the 4 directions) is selected.

- 4 sub-problems are produced. Each contains 1 subtree connected to the central edge and all leaves that are a part of the short quartet.
Step 2: Tree improvement

Simply a tree search on a smaller tree

DCM is a “meta-method” that can be used with almost any type of large-scale tree inference.
Step 3: Tree Merge (Supertree analysis)

The step of “glueing” the trees for subproblems together is a supertree analysis.

If there is no conflict between the input trees, the problem is trivial.

Roshan et recommend using a Strict Consensus Merger - collapse the minimal number of edges required to make 2 trees display the same tree (for the leaves that they have in common).
\[
\begin{align*}
\text{Diagram 1} & \quad + \quad \text{Diagram 2} \\
= & \quad \text{Resulting Diagram}
\end{align*}
\]
Step 4: Tree Refine

Optional step - some tree searching methods require binary trees

Step 5: Tree Improve

Another “base method” tree search (but with a large set of taxa, so the search often has to be less thorough)
Decompose
Decompose

Small Tree

Improve
Decompose Tree Merge

Small Tree Improve

Decompose

Small Tree

Tree Merge
A recursive algorithm is one that calls (invokes) itself.

A definition of the function to compute the factorial is the classic example:

```python
def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n - 1)
```

Recursion is often used when it is easy to perform a few tasks, but then you are faced with the same problem you originally faced, but on a smaller scale.

Recursive DCM3 arises from the recognition that, when we break our full set of taxa into subsets some of them may still be too large for thorough searching. We can use another level of DCM to break them down into smaller problems.
Iteration

Because the decompositions are sensitive to the starting tree, we may do a better job decomposing the tree into closely related subtrees if we have a better estimate of the tree.

So we can simply repeat the whole recursive DCM process