Lecture 8 – Searching Tree Space

I. Introduction: There are many uses of phylogenies (I would say most) for which a "good" tree isn't good enough, and we really are interested in (at least trying to find) the best tree under some optimality criterion.

Papers published in the phylogenetics literature almost always include phylogenies produced using some type of search of the vast tree space, under at least one (often two or three) of the optimality criteria we've discussed.

These searches range from exact searches (very rarely), that guarantee that we'll find the optimum topology(ies), to heuristic searches, that aim to search only a portion of the tree space. Heuristic searches traditionally work by hill climbing, and therefore are not guaranteed to find the optimal tree(s); that is, they are susceptible to becoming trapped on local rather than global optima.

We can think of the problem in terms of tree space, with the array of possible trees spread out in two dimensions and the third dimension being optimality. So each tree has an optimality score, and trees are linked by a single branch swap (which we'll examine in a few minutes).

There may be local optima, regions of tree space that contain good, but not best trees. These are called tree islands.

The challenge is to traverse tree space, avoiding these local optima, and find the global optimum. The degree to which there are multiple peaks in this tree landscape determines how difficult a task this is.

II. Exact Searches – There are two ways that we can guarantee that we'll find the globally optimum tree.

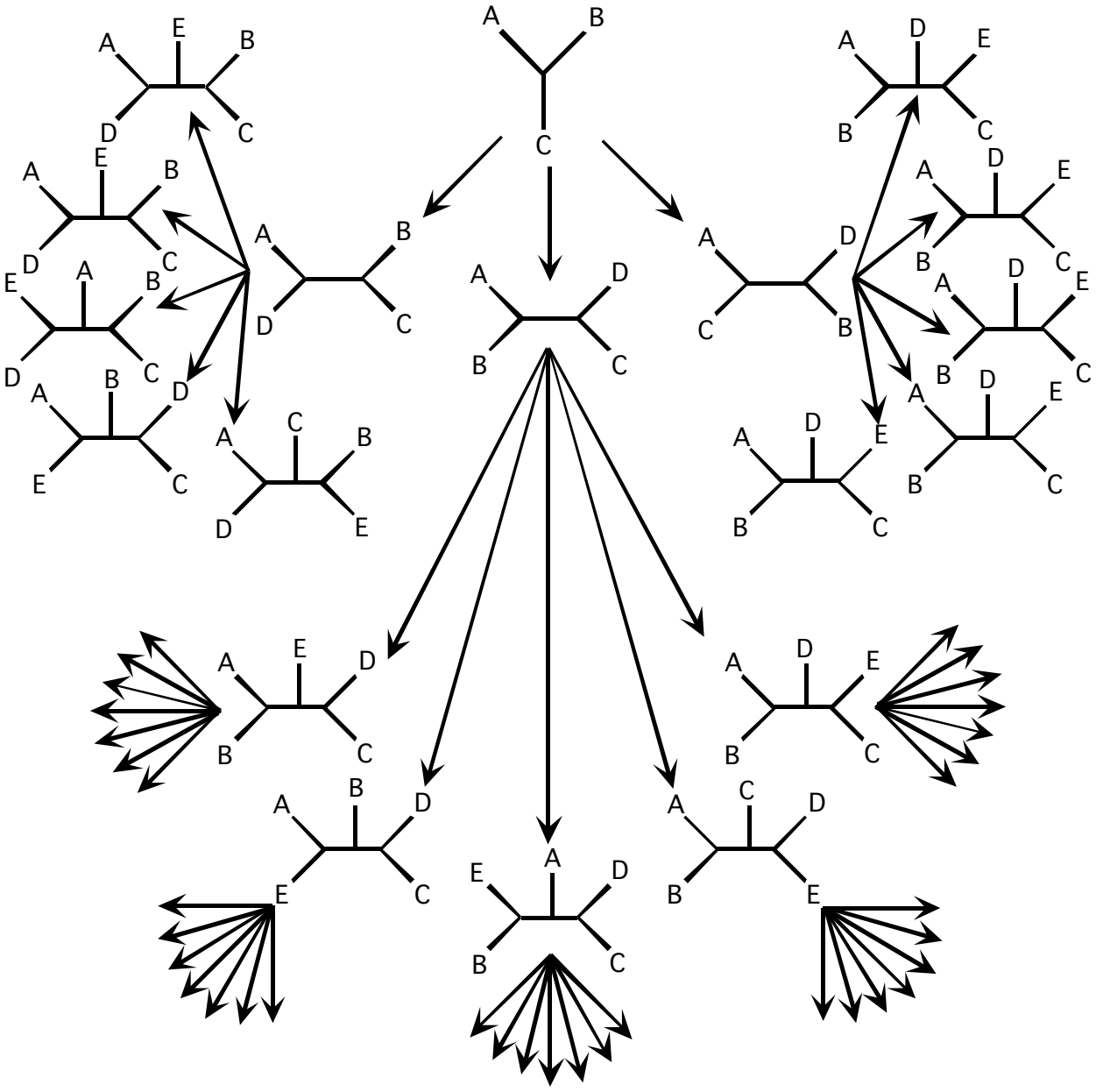
A. Exhaustive Searches

The most straightforward manner in which to do this is to look at all possible trees. Doing so will hit all optimality peaks, and we are guaranteed to find the global optimum.

Obviously, this is limited to small data sets, but understanding how an exhaustive search works will facilitate our discussion of other types of searches.

Exhaustive searches first require a means of generating all possible topologies for a particular set of taxa.

This is accomplished via a **search tree**. We've already hinted at what a search tree is in our look at stepwise-addition algorithms.



This is a central concept in searching for optimal trees because it conveniently defines tree space and provides a way to traverse it.

The structure of the search tree also suggests a more efficient exact search.

B. Branch & Bound Searches

This method uses the search tree, and first follows some path until all taxa are added.

The score of the tree is then stored as the current bound, and another path through the search tree is traversed.

As soon as the current bound is exceeded, passage out the current branch of the search tree is halted. There's no way that proceeding out the path (i.e., adding taxa) will shorten the tree (if the criterion is parsimony), so no phylogeny farther out that part of the search tree can be optimal.

This approach allows one to conduct an exact search, without having to examine all possible trees. The approach is viable for up to around 17 - 18 taxa for parsimony and clean data.

III. Heuristic Searches – As I've mentioned, for most applications we're forced to search for optimum trees using some type of heuristic search.

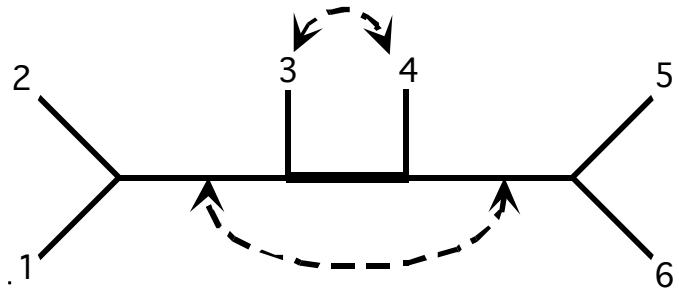
These are not guaranteed to find optimal trees.

The most common implementation of heuristic searches generates a starting tree with one of the algorithmic approaches (usually stepwise addition), and then tries to improve on this tree by a series of local rearrangements.

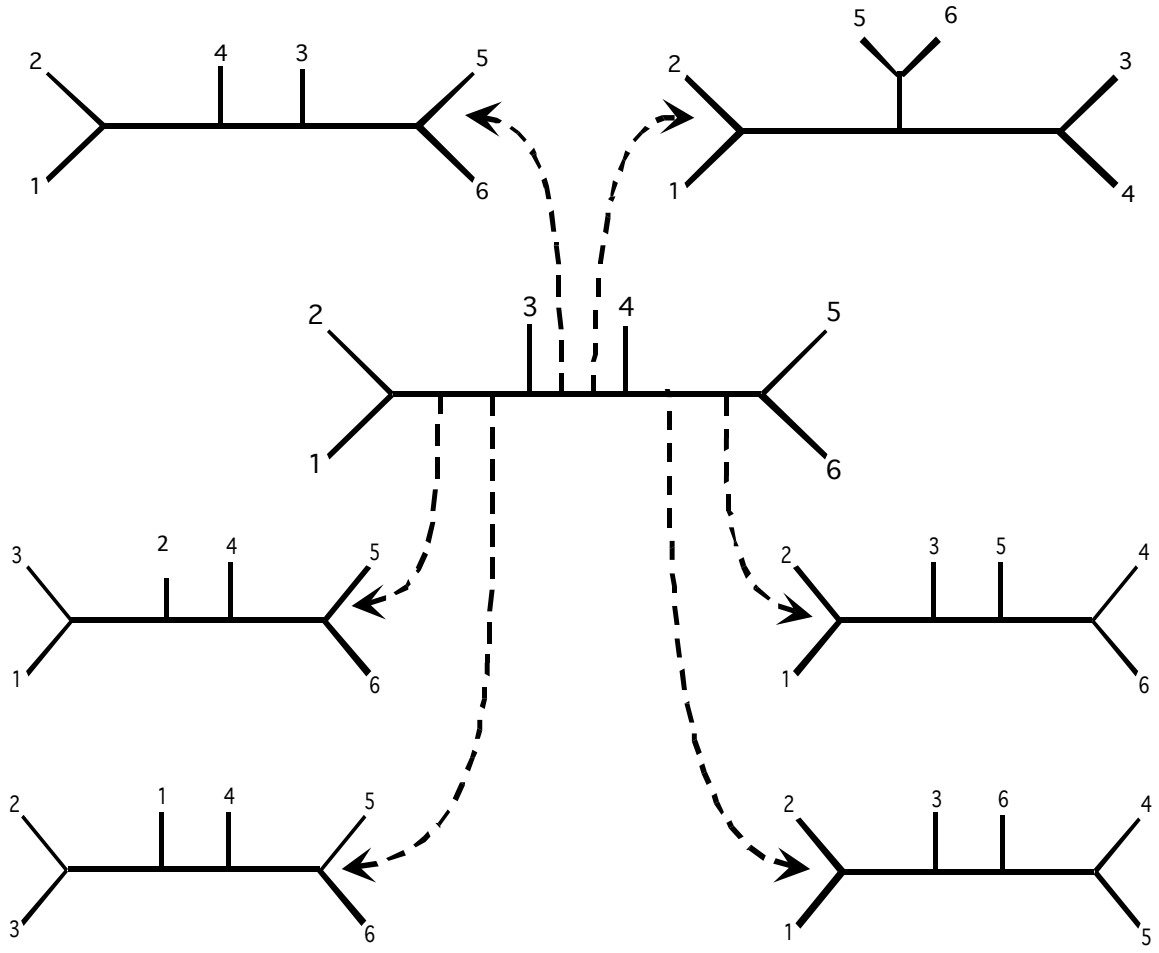
Rearrangements are accomplished via **branch swapping**, and there are a number of approaches. We'll look at three, from least to most rigorous.

A. Nearest-neighbor interchange (NNI) is the least rigorous swapping approach.

Consider the six-taxon tree show below:



There are two possible NNI rearrangements per internal branch. These are indicated by the arrows for the thick middle branch. For a tree with n taxa, there are $n - 3$ internal branches. Thus, there are $2(n - 3)$ NNI rearrangements for any tree.



This figure shows all 6 possible NNI rearrangements of this tree.

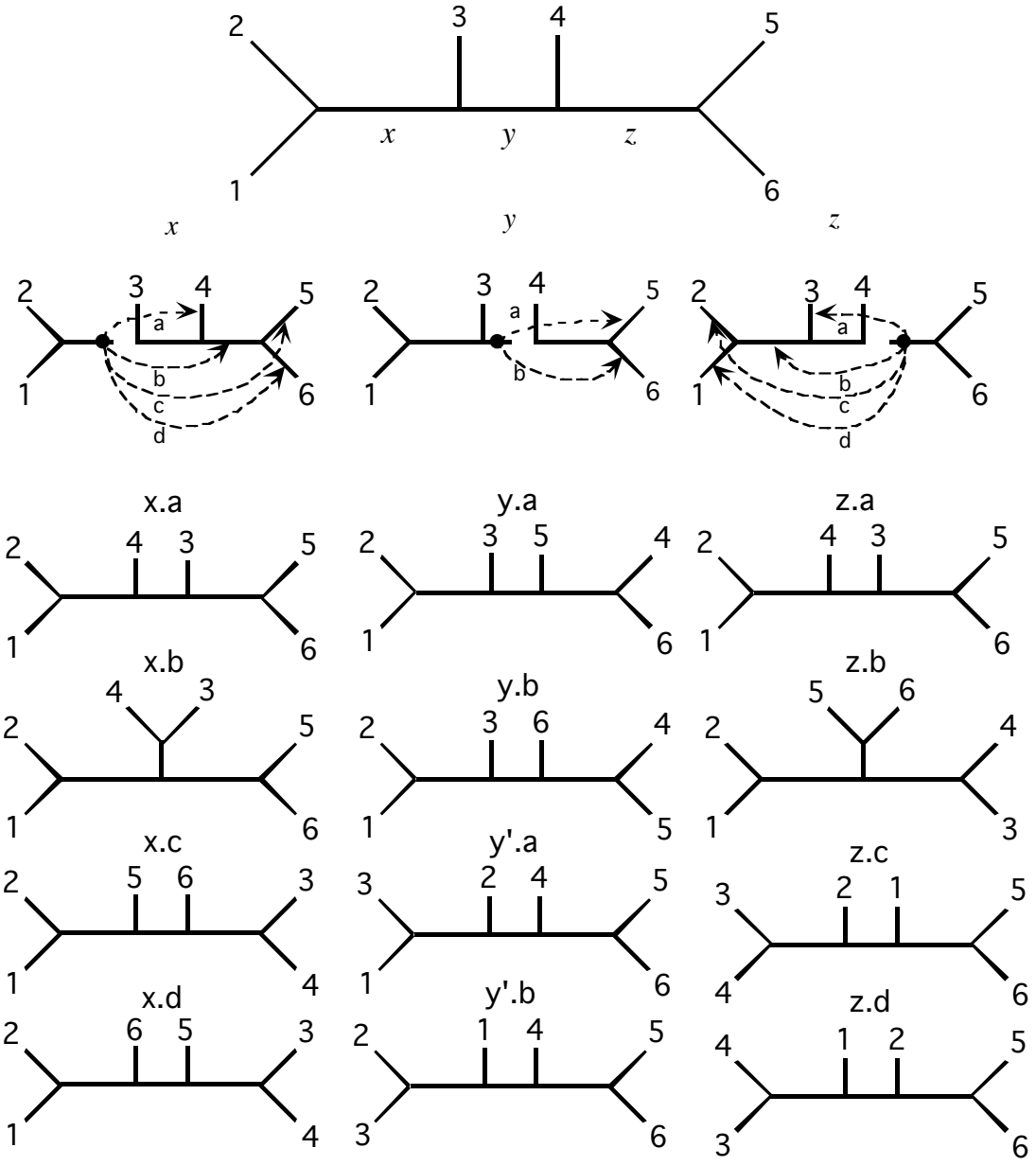
Note that, for six taxa there are 105 possible topologies, and a greedy NNI search might only examine 7 (the starting tree plus all six rearrangements).

B. Subtree Pruning-Re-grafting (SPR) is a more rigorous swapping scheme.

This method operates by breaking all branches to make subtrees and grafting the subtrees onto each branch of the remaining trees. Consider again the same 6-taxon tree:

Here, we have labeled the internal branches x, y, & z.

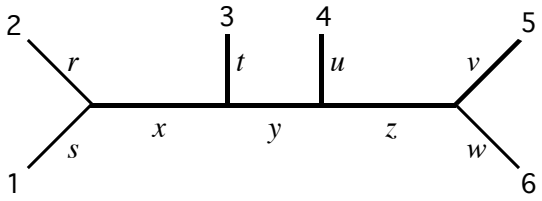
The idea is to generate subtrees by breaking each branch, and then re-grafting the subtrees in all possible ways:

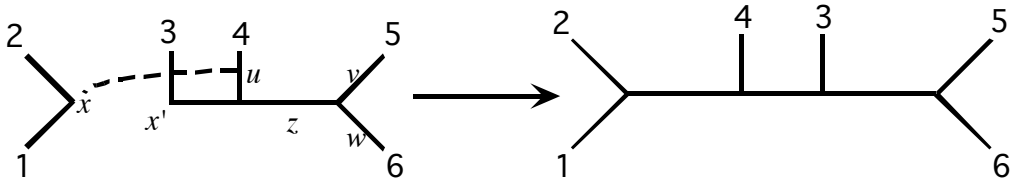


This shows the SPR rearrangements that result from pruning branches x, y, & z.
In addition, each of the terminals would be pruned and regrafted to all possible branches.

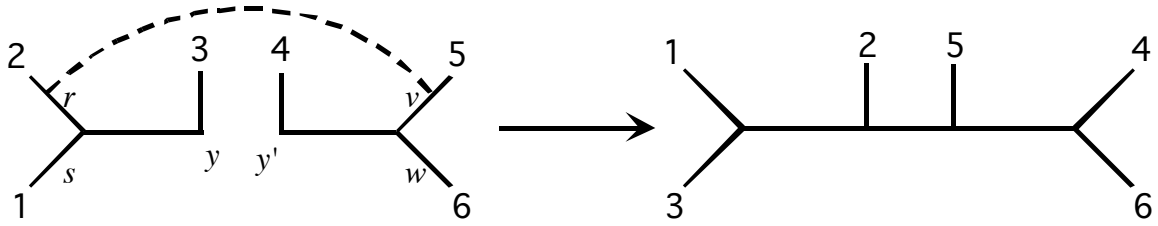
Thus, there will be $4(n-3)(n-2)$ SPR rearrangements. For this six-taxon tree, there would be 48 rearrangements of 105 possible trees, so SPR searches tree space pretty well for 6 taxa. However, some of the 48 rearrangements are redundant, so we aren't actually looking at 48 unique trees.

C. Tree bisection-reconnection (TBR) is the most rigorous strategy for branch swapping.





If we bisect the tree at branch *x* and reconnect to branch *u*, we would get the tree shown. Reconnections would also be made to branches *z*, *w*, & *v*.



If we bisect the tree at branch *y* and reconnect to branch *v*, we would get the tree shown. Reconnections would also be made from branch to branches *y'* & *w*, from branch *s* to branches *y'*, *v*, & *w*, and from branch *y* to branches *v* & *w*.

All branches are bisected, and reconnected in all possible ways. It's not possible to generalize how many TBR rearrangements could be made for a tree of a given size (as we could with NNI & SPR), but TBR swapping searches tree space more thoroughly than SPR or NNI.

So, of the three commonly applied branch-swapping strategies, NNI is the least rigorous, SPR is next, and TBR is the most rigorous.

By building a starting tree and hill climbing from it, we can conduct a heuristic search of tree space.

D. The next question of **how greedy should we be**?

Let's use as an example a smallish 26-taxon data set,

First, let's be very greedy.

It's common to encounter ties during tree building. During the stepwise addition of taxa to build a starting tree, I only saved one tree per step (arbitrarily the first of the shortest options).

Furthermore, we can encounter ties during branch swapping. That is, we may swap to a new tree that has the same score we're swapping. A greedy approach will ignore that, and only reset the tree for swapping if we find a better tree. I only swapped on the single shortest tree I've saved.

NNI, I'll look at 42 trees (if I ignore ties).
SPR, I'll look at 2072 trees (if I ignore ties).
TBR, I'll look at 5816 trees (if I ignore ties).

We can be far less greedy by saving multiple trees at each step in stepwise addition (and therefore using more than one starting tree for swapping), and by saving all possible optimal rearrangements for branch swapping.

So now, NNI: 140
SPR: 6212
TBR: 16,604

In this case, I found the same trees as the more greedy approaches, but that's often not the case.

E. Avoiding Local Optima with Random Addition Sequences

Due to the hill-climbing nature of these heuristic searches, they are prone to getting stuck in local optima. These are tree islands that don't contain the globally optimal tree(s).

The most common thing that is done to check for tree islands is to use the addition-sequence dependence of step-wise addition in building starting tees.

We mentioned earlier that the order in which we sequentially add taxa in step-wise addition can have an influence on the tree that is built.

If these different starting trees occur on different tree islands, we may be able to avoid becoming trapped on a local optimum.

Typically, this is done by using random addition sequences to build the starting trees, and swapping is conducted to completion on each. We may use 100 random addition sequences, or as many as practical.

So in the above example, using the least greedy strategies and using starting trees generated by 100 random addition sequences, we'll look at 341,355 different trees.

Island	Size	First tree	Last tree	Score	First replicate	Times hit
1	2	1	2	278	1	99
2	1	-	-	279	97	1

So there are two islands, one with two trees of 278 steps and one with one tree of 279 steps.

This approach is usually used for data sets of up to around 100-200 taxa.

IV. Newer Approaches for Large Datasets (which are becoming the norm).

These are based on the idea that we may be better off spending less effort searching one island and more effort searching for multiple islands.

A. Simulated Annealing

In the mid 1980's, computer scientists developed Simulated Annealing, a method of optimization designed to search a large, complex, discrete search space.

It applies quite well to phylogenetics because these are exactly the properties of tree space.

Laura Salter Kubatko was one of the first to apply it to phylogenies as a means of estimating ML trees (Salter and Perl, 2001. Syst. Biol. 50:7).

It applies an MCMC approach to searching tree space and stochastically permits down-hill moves across tree space.

Steps:

Generate an initial state (i.e., a starting tree). The early implementations used a random tree; recent implementations build a starting tree by stepwise addition.

Propose a stochastic change to the initial state (usually a minor change). This was initially derived via a random NNI, but bigger swaps are now allowed.

If the proposal improves the tree (i.e., has a better ML score), the move is accepted.

Proposals (NNIs) that degrade the tree are accepted with a small probability that is proportional to how much worse the proposed tree is than the current tree.

Some stopping rule is enforced.

The idea of simulated annealing is to alter the acceptance probability over the course of the MCMC run.

In early generations, the probability of accepting a worse tree is relatively high. This provides a good chance of moving between tree islands early on, in the hopes that we'll find the globally optimum island

As the chain runs longer, the probability of accepting a worse tree is decreased, and the rate of change is called the **cooling schedule**. Thus, later in the chain the run is behaving increasingly like a hill-climbing approach.

B. Commonly Used Approximate Searching Programs - RAxML

Stamatakis (2005. Proc. of IPDPS2005) has implemented a modified simulated annealing in his RAxML package.

First, RAxML starts with MP trees generated by stepwise addition and several random addition sequences.

Second, "lazy" SPR is used to propose new trees that results in enormous time-savings.

There's a limit in the number of re-grafts that are examined (only fairly close to original tree).

Then, branch lengths are optimized only for the three branches adjacent to the re-grafting point. **New trees that decrease the likelihood are sometimes accepted early in the run**.

Third, RAxML builds proposals to alter branch lengths and model parameters that are only accepted if they improve the likelihood (i.e., this aspect of the searches are entirely hill climbing).

This approach allows pretty thorough searches of tree space really quickly, which permits us to estimate ML trees for very large data sets (e.g., a few thousand taxa). It has become a preferred approach to ML-tree estimation. IQTree (Nguyen et al. 2014. Mol. Biol. Evo. 32) is similar.

C. Commonly Used Approximate Searching Programs: IQ-TREE

Candidate set of (100) starting trees is generated via parsimony using stepwise addition with random addition sequences.

The likelihood score of each is calculated and the 5 best (ML) trees are retained.

Hill climbing is conducted on each of these with local NNI (under ML), reoptimizing only neighboring branches; the 5 best (ML) trees are saved

One is selected at random for stochastic NNI; select internal branch at random for NNI swapping. Add perturbed tree to set of 5 if it's better than any and start over.

Repeat until 100 random perturbations have not found a better tree.