

## Lab 1 Lecture Notes - Searching Tree Space

**I. Introduction:** There are many (perhaps most) uses of phylogenies 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.

These searches range from exact searches, 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 work by hill climbing, and therefore are not guaranteed to find the optimal tree(s) (i.e., they are susceptible to becoming trapped on local rather than global optima).

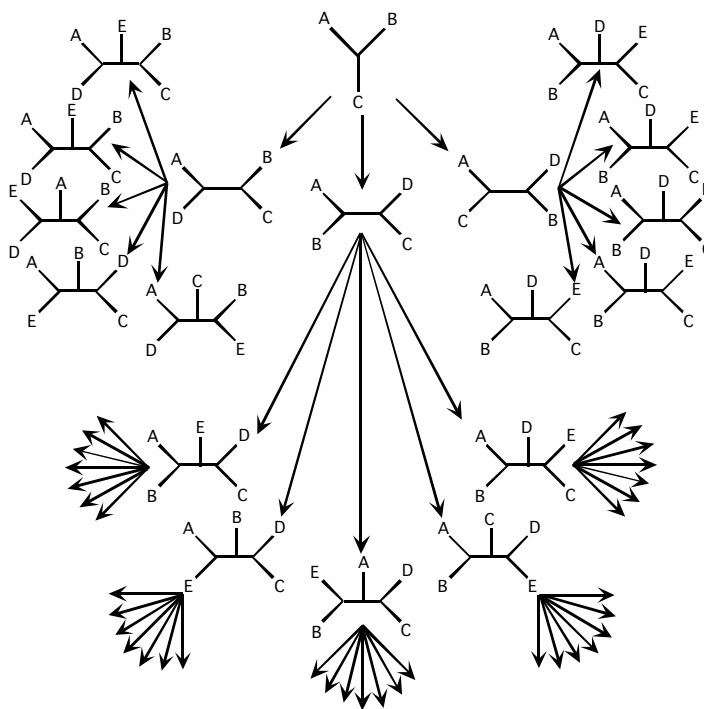
**II. Exact Searches** – There are two ways that we can guarantee that we’ll find the globally optimum tree.

### 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**.



## B. Branch & Bound Searches

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

The length 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 14 - 15 taxa.

**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.

### A. Stepwise Addition Methods

Other algorithms to build trees take the opposite approach. They add taxa sequentially rather than decomposing a star. Some older distance methods took this approach (e.g., the distance Wagner approach), but these are almost never used anymore (I published one in my first paper ever), and have been supplanted by NJ.

Nevertheless, Stepwise Addition is incredibly important in modern phylogenetics because of its role in searching through tree space under optimality based searching methods.

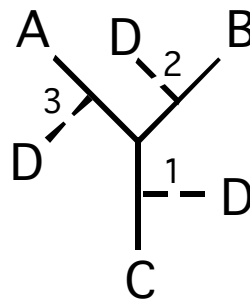
The basic idea is to start with three taxon tree, compute its score under some criterion and add a fourth in the manner that incurs the smallest decrease in optimality (e.g., smallest increase in length).

I'll use a parsimony example that's easy to optimize (from Felsenstein, pp. 2 & 53).

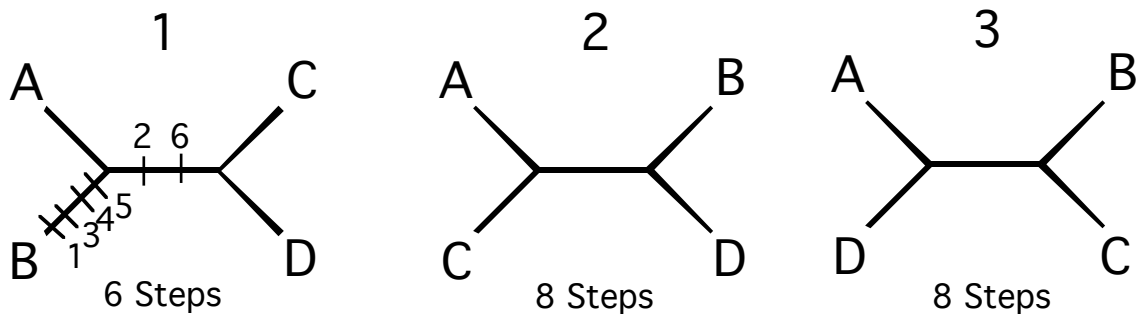
	Characters					
	1	2	3	4	5	6
A	1	0	0	1	1	0
B	0	0	1	0	0	0
C	1	1	0	1	1	1
D	1	1	0	1	1	1
E	0	0	1	1	1	0

First, we pick three taxa. We'll use the order the taxa appear in the matrix as our input order. This matters, because the method is starting-point dependent.

So for A, B, & C there is only a single unrooted tree, shown by the solid lines.



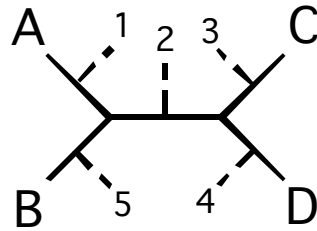
There are 3 possible places to add taxon D, indicated by numbers 1 – 3. The optimality score (tree length, in this example, using either the Fitch or Sankoff algorithm) is computed for each of these:



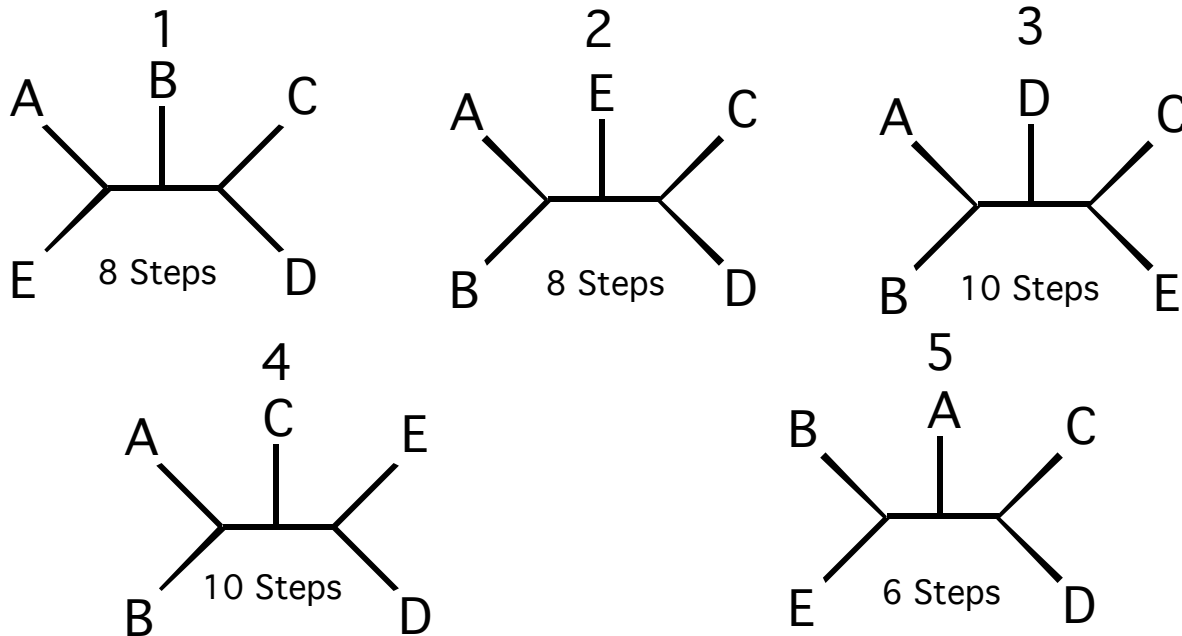
Option 1 is chosen because it has the shortest in length (i.e., is optimal).

Taxon E is then added to tree 1 and there are 5 possible placements for a new taxon (because there are 5 branches in the tree at this point).

These are labeled 1 – 5, and again, are indicated by the dashed lines:



Again, characters are optimized on each of these 5 possibilities:



Here, we've used stepwise addition build a tree(s) using parsimony as an optimality criterion. We could have used any criterion wish (so long as the data are amenable).

Now, if we had a 6<sup>th</sup> taxon to add, we would consider all 7 possible placements.

Thus, unlike in star decomposition, the number of calculations increases as we build a tree via stepwise addition.

**Addition sequence** is critical. The method is starting-point dependent, so different addition sequences can produce different stepwise addition trees.

Options:

As is – In the order that the taxa appear in the matrix

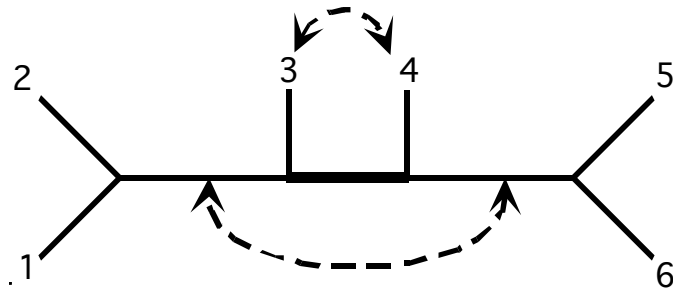
Shortest – One may check all triplets, chose that which has the shortest tree and, at each step, assess all candidates, and choose that which adds least length.

Random – One may use random addition sequences and replicate a number of times. For clean data, there should be only small differences in the stepwise addition tree, but for most real data there will be multiple stepwise addition trees. We'll make use of this in searching tree space.

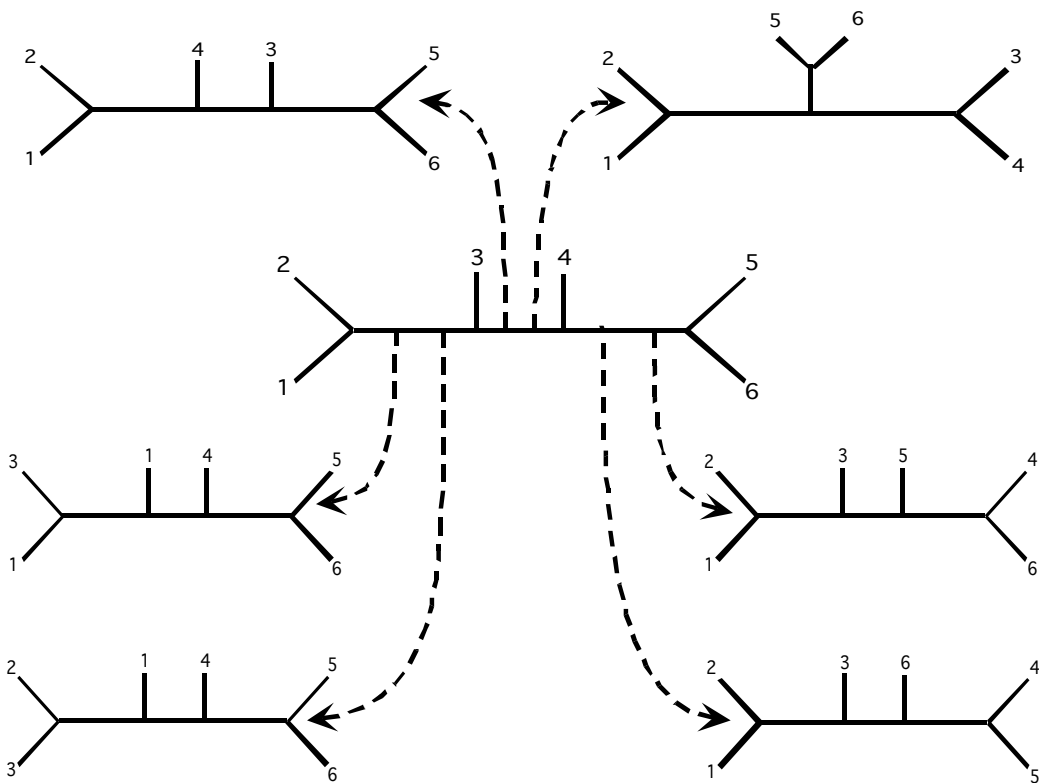
## B. Branch Swapping

1. **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.



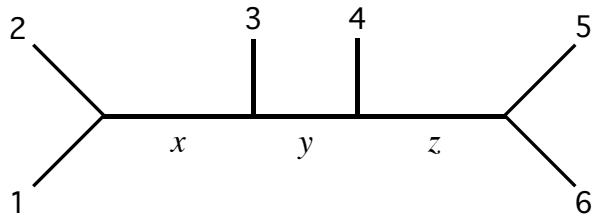
This figure shows all 6 possible NNI rearrangements of this tree.

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

**2. Subtree Pruning-Regrafting (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*.



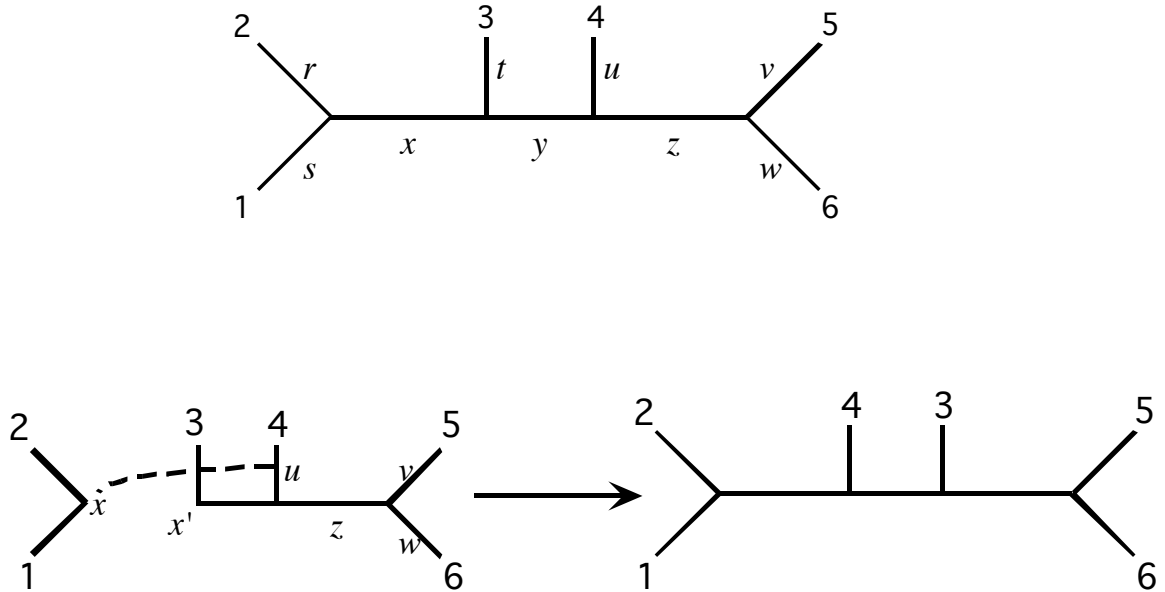
The idea is to generate subtrees by breaking each branch, and then regrafting the subtrees in all possible ways:



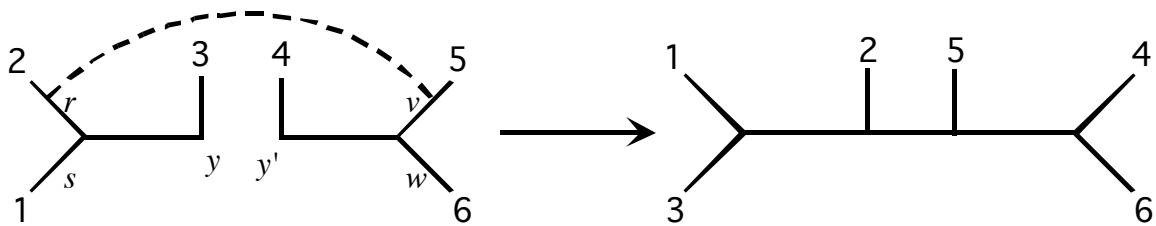
This shows the SPR rearrangements that result from pruning branches *x*, *y*, & *z*. **In addition, each of the terminals** would be pruned and regrafted to all possible branches.

Thus, there will be  $4(n - 3)(n - 2)$  SPR rearrangements. For this six-taxon tree, there would be 48 rearrangements. For 6 taxa, there are only 105 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 different trees.

**3. 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.