

Biology 545 – Lab Exercise 1 – Parsimony and Introduction to PAUP*

This exercise has three goals. The first is to provide you with some experience using PAUP*. This program is written by Dave Swofford, one of the most influential systematists in the world. I'll admit that I'm biased, but I think this is the most important and flexible (and possibly widely used) program in phylogenetics. The second goal is to teach you how to conduct parsimony analyses (regardless of program), and the third is to illustrate the effects that decisions in search strategy and character weighting can have on parsimony estimates of phylogeny.

One reason PAUP* is so useful is that it has the ability to evaluate trees under each of the three optimality criteria we'll be discussing. In addition, it has the flexibility to run both quick & dirty "algorithmic" analyses (such as neighbor joining, quartet puzzling, UPGMA, etc.) as well as searches of tree space (both exact and heuristic). Therefore, it's important to get to know how to use PAUP*. In addition, PAUP* is available for essentially any platform. Its original manifestation was as a DOS program (in the 1980's), but it really came to dominate phylogenetics in the early 90's, when it was only available for Macs. If you're used to using PAUP* on a Mac running classic OS, you'll have to relearn a lot of stuff. The reason is that the classic OS version can run everything from pull-down or pop-up menus; these are either not extensive, in the Windows version, or not available, in the various Unix versions.

For tonight, we'll restrict our attention to parsimony analyses that can be conducted using PAUP*. In order to proceed, you'll have to get the data set from the course web site: <http://www.uidaho.edu/~jacks/Biol545ParsDat.nex>. Your browser will display a nexus file that you can cut and paste into any text editor. Once you've done this, you can open it with PAUP* by typing:

```
paup file_name
```

You'll have two options; you can run in interactive mode (where commands are typed into the command line), or you can run in batch mode (in which commands are either placed in a paup block in the data file or in a separate command file). I don't care how you choose to proceed; I often do both. If I'm running relatively short jobs, I typically use interactive mode. If I'm running a medium (overnight) to long (several days) job, I usually use batch mode and include save and quit commands. This prevents lab mates from quitting completed runs before results are saved.

Although there is no extensive user's manual, there are excellent help documents freely available (in pdf format) from: <http://paup.csit.fsu.edu/downl.html#Anchor-58521>.

Exercise 1 – Search strategy

In this exercise, we'll demonstrate the need to search tree space thoroughly.

First, we'll do a very greedy search, by only saving one tree at any time and using NNI swapping.

Type the following commands.

```
set maxtree=1 increase=n;
```

```
hs swap=nni;
```

[The first line limits the program to a single tree in the tree buffer, and prohibits the program from increasing that limit if > 1 equally parsimonious tree is found, either during construction of the starting tree or branch swapping. The second line tells PAUP to conduct a heuristic search with branch swapping conducted using nearest-neighbor interchanges. With the default settings, the starting tree is attained by stepwise addition with simple addition sequence.]

Note the length of the tree found.

Next, we'll do a moderately rigorous search using the defaults. Type the following.

```
set maxtree=100 increase=prompt;  
hs swap=tbr;
```

[Here, we'll allow PAUP to begin by storing up to 100 trees in the tree buffer, and allow the program to increase that limit as needed (if > 100 equally parsimonious trees are found). Again, the second line initiates a heuristic search, but now with branch swapping conducted using tree bisection and reconnection. Again, we're only using a single starting tree, generated by simple stepwise addition.]

Again, note the length of the tree found. Is it different? This represents the level of rigor in a one would get simply running a parsimony analysis with PAUP* at its **default settings**. In this search, we kept multiple equally parsimonious trees at every stage, but only the stepwise addition tree(s) generated from a single addition sequence as the starting tree(s) for branch swapping.

Now, we'll search the tree space a little more rigorously. Type the following:

```
hs addseq=rand nrep=500;
```

[Since maxtree, increase, and swap are persistent settings, they're the same as above. Now we're conducting 500 replicate heuristic searches, each starting with a stepwise addition tree generated using a different random addition sequence].

First, did you end up finding a better tree? Second, check out the large number of tree islands in this data set. For the best island you found, how many addition sequence replicates hit it? What do you think that indicates about the confidence that you've found the globally shortest tree?

Exercise 2 – Character weighting

Now I want you to explore some of the weighting strategies we'll discuss in lecture. First, save the tree that you estimated from equal weights:

```
savetree file=EqualWts.tre;
```

I want you to assign different weights to each of the codon positions; try the 2,5,1 scheme that I'll mention in lecture. I've already created character partitions in the data file that reflect codon structure by placing the following assumptions block after the data block:

```
begin assumptions;  
  charset 1stpos = 1-727\3;  
  charset 2ndpos = 2-727\3;  
  charset 3rdpos = 3-727\3;  
end;
```

You can now use these character partitions to weight the positions differentially (i.e., use non-uniform w_i 's). Type:

```
weights 2:1stpos;  
weights 5:2ndpos;
```

Since the default weight is 1, you don't have to do anything to 3rdpos.

Now run a heuristic search. If you haven't quit PAUP since the last search, the search settings should still be TBR branch swapping and 500 replicate searches with random addition starting trees.

You can always check the current setting by typing the command followed by a question mark:

```
hs ?;
```

So do a search:

```
hs;
```

Now save the trees to a file called CodonWts.tre

Are these trees different than the equal weights trees?

Reset all characters to equal weights (type `weights 1:1stpos; weights 1:2ndpos;`)

Now we want to use a step matrix to down-weight transitions. Again, we'll use the assumptions block to define step matrices. PAUP allows us to specify step matrices by defining "usertype"s. I've written three different step matrices to give transversions more weight than transitions.

```
usertype 1 = 4  
  G  A  T  C  
  0  1  10 10  
  1  0  10 10  
 10 10  0  1  
 10 10  1  0  
;
```

You can call these by typing:

```
ctype 1:all;
```

This tells PAUP to apply the step matrix called 1 to all characters.

Now do a search with each of the three step matrices (but only do 100 replicate random addition sequences). Save the tree from each one to a file.

Now I want you to compare these trees with the ones you estimated with differentially weighting characters.

You can do this using `gettrees` & `treedist` commands we used in the alignment exercise.

```
gettrees file= EqualWts.tre mode=7;  
gettrees file= CodonWts.tre mode=7;
```

etc. [mode=7 keeps all trees in buffer and in the file].

Just hand in the distribution of symmetric-difference distances.

Finally, I want you to combine one of the step matrices with codon-position weights; 100 random addition sequences will suffice. This represents a rather complex weighing scheme, but we could apply a different step matrix to each character partition (i.e., codon position) if we wanted to.

What happens to tree space when we use both non-uniform transformation costs and differential weighting across characters?