Lab 06: Support Measures
Bootstrap, Jackknife, and Bremer

So you ran a heuristic search and got a tree. Now what? How do you tell how well (or poorly) supported the tree you've come up with is? There are various different ways to get a sense of how robust your data is –that is, is my tree just a fluke? Or, given the data that you have, are very few other trees possible? The goals for this lab are for you to use a test tree to perform each kind of support analysis, understand how they work, and be able to use whichever ones you choose in your own final project. Today we'll go ahead and use parsimony for all our searches, but you can also use the support measures with distance (which will be very quick) or likelihood (which may take a long time).

Exercise 1: Build a Test Tree

Open PAUP*.

Create a new folder (eg. IB200_Lab6) and then set this as your working directory for the day.

```
paup> cd /Users/yourusername/Desktop/IB200_Lab6
```

Load your data into PAUP. Either use some of your own data (in Nexus format) or use the sample file primate-mtDNA.nex Remember PAUP needs the path of where this file is, so dragging and dropping the file generally works.

```
paup> execute filename
```

Change the optimality criteria to parsimony (actually this is the default in PAUP, so this step isn't really necessary).

```
paup> set criterion = parsimony
```

Then, run a heuristic search:

```
paup> hs
```

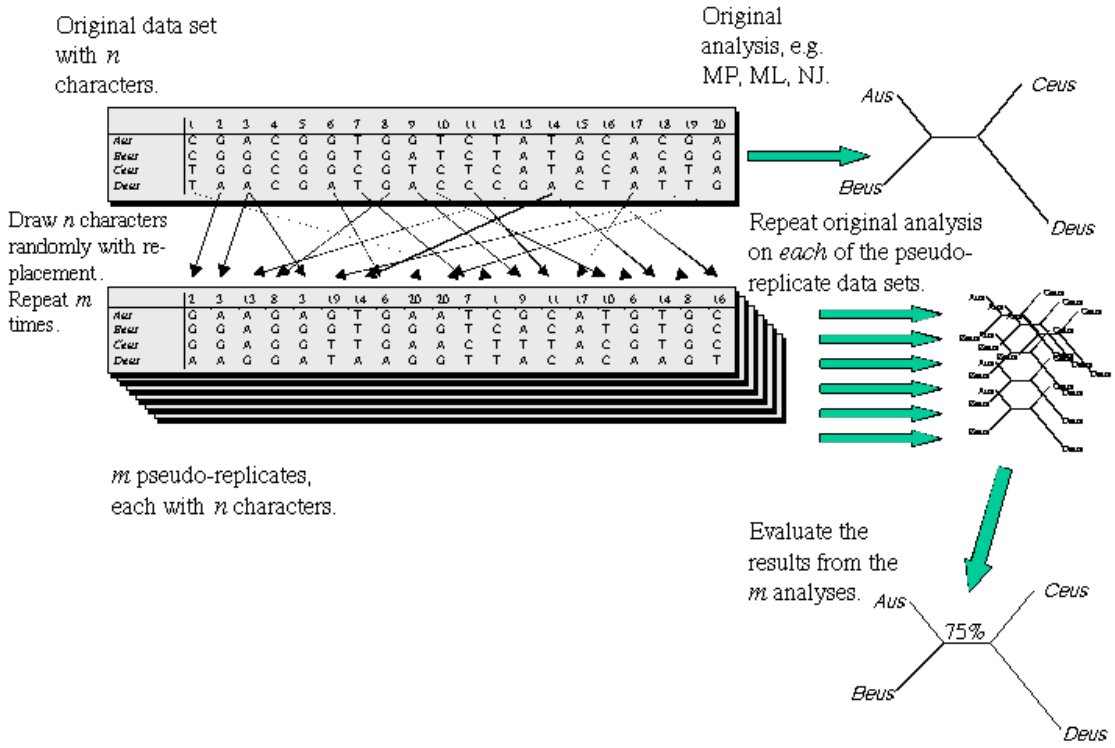
Then save your trees:

```
paup> savetrees file=parstree.tre
```

Exercise 2: Estimate Support by Bootstrapping

Okay, now let's figure out how well supported these groupings are. One measure of support is called the **Bootstrap**. Here's how it works: It chooses columns randomly from the matrix until is has chosen the same number of columns as were in the original matrix. Because it returns to the original matrix each time it chooses a new column, some characters may be

represented several times in the bootstrap matrix, while others are omitted. This is known as resampling the data with replacement. In practice, although it is possible to randomize taxa, bootstrapping almost always randomizes characters. Bootstrapping calculates a support value for each node based on the fraction of samples that support that node. Bootstrap provides a number on each node (0–100%). The highest support value is 100, while values below 70 are usually considered weak. Values below 50 aren't shown. In fact, branches below 50 are collapsed and shown as a polytomy. Bootstrap support is somewhat sensitive to the number of replicates used, but not terribly so.



[Image from http://artedi.ebc.uu.se/course/X3-2004/Phylogeny/Phylogeny-Credibility/Phylogeny-Credibility.html]

To run a bootstrap analysis with a hundred replicates, using a heuristic search for each replicate, and randomizing the order of taxa while building the trees on your data, type:

```
paup> bootstrap nreps=100 search=heuristic /addseq=random;
```

See the PAUP manual for discussion of other options, including the ability to save all your bootstrap trees. It will take a moment to run, longer than the heuristic search because it is, in effect, running 100 heuristic searches. Still, for this data set it doesn't take took long at all. When it's done, PAUP will display a 50% majority rule consensus tree from all 100 bootstrap runs. [This type of consensus tree only shows clades that are present in more than 50% of the primary trees.]

Question #1: **Does the bootstrap support all the nodes that appeared in the Maximum Parsimony tree? At which nodes does it differ?**

PAUP will also display a list showing the frequency of different taxon bipartitions (that is, how often out of the 100 replicates the taxa were grouped together.) If you look at the tree, you'll see that Pan has a (3) after it and Gorilla has a (4). Look down the chart until you see a line where there is a * under 3 and 4, but not under any of the other numbers. The frequency on this line is 55.00, which means these two taxa were grouped together in about 55 out of the 100 runs. (You'll notice some of these numbers are fractional because some runs produced more than one most parsimonious tree.)

Ok, now you may want to export the fancy new bootstrap values you just came up with. Well, luckily there is a way to do that:

```
paup> savetrees from=1 to=1 savebootp=nodelabels file=filename.tre;
```

Note that you must supply the tree numbers that you want to save (here, from 1 to 1, since there is only one bootstrap tree) and the file name you want to save to. You can view these node labels on the tree in FigTree or Mesquite.

Question #2: **Figure out a way to show your tree with bootstrap support in FigTree [Note: The program can be a bit glitchy for some reason so you may need to open the program first and then go to File → Open to select your file]. Highlight the weekly supported 'Pan'-'Gorilla' clade, take a screenshot, and send it to me.**

Exercise 3: Estimate Support by Jackknifing

Jackknifing is very similar to bootstrapping, but rather than resample the data, it uses subsets of the data. (This is also described as resampling without replacement to create a smaller dataset.) The purpose of this is to see if excluding certain characters has a big effect on the shape of the tree. (You can imagine that some "outlier" character might have a disproportionate influence on the relationships that are reconstructed; jackknifing is an attempt to get around this.) For whatever reason, it is much less common in the literature than bootstrap support. Still, people do use it from time to time so it is good to know what it is.

```
paup> jackknife nreps=100 search=heuristic /addseq=random;
```

Export the jackknife values the same way you exported the bootstrap ones:

```
paup> savetrees from=1 to=1 savebootp=nodelabels file=filename.tre;
```

Question #3: **Open your jackknife tree file in FigTree. That many significant digits is unnecessary so try to get rid of them. Then highlight the weakly supported clade with 'Pan', take a screen shot, and send it to me.**

Exercise 4: Estimate Support by Bremer Support/Decay Index

For those that eschew a statistical view point (or who like to use a few different frameworks), a good alternative is to determine whether a group of interest occurs in other trees that are almost equally short. Another way to this about this is to consider that with every tree search, the heuristic must make multiple decisions about which characters are true homologies and which must be homoplasies. Generally, the grouping that leads to the most homologies is used. Bremer support asks whether there are other ways to analyze the homoplasious characters that lead to trees that are only a few steps longer. As a rule of thumb, a Bremer score of 3 is good and a score of 5 is "highly supported."

PAUP doesn't calculate Bremer support directly, so you have to use some tricks to get these numbers. I will show you how to generate Bremer support numbers using the program TreeRot.

TreeRot

1. Place your initial parsimony tree file (parstree.tre) within the TreeRot folder.
2. Double-click the TreeRot.sh icon (if that doesn't work, open another command line window and type: perl [drag and drop TreeRot.pl]
3. Enter 1 to generate a PAUP command file. Follow the prompts to enter
    a. The name of the tree file (parstree.tre)
    b. A name for the command file that will be generated (parstree.constraints)
    c. A name for the PAUP log file (parstree.results)
4. Answer NO to partitions.
5. Return to PAUP. Execute the command file from TreeRot (parstree.constraints)
6. The file parstree.results will be saved to your current working directory, so make sure to put that into the TreeRot folder for the next step.
7. Launch TreeRot again. Enter 2 to parse the PAUP log file.
8. Enter the name of the log file (parstree.results). Name the file for the parsed results (parstree.summary)
9. Open this file in FigTree

***Question #4*: Show your Bremer support values in FigTree. Increase the size of these numbers so I can see them. Take a screen shot and send it to me.**

You can also write a script using MacClade, which will generate Bremer numbers based on a set of constraint clade analyses (see MacClade manual) or directly at the command line. If you are curious, here is the fairly tedious way of doing this manually:

1. Copy the anolis.nex file to the folder on your desktop (from the website or from the MrBayes folder.) Or, you can use your own data again.
2. Execute the file in PAUP
3. `set criterion=parsimony;`
4. `hs;`

Once the heuristic search is done, PAUP will output a little paragraph that looks like this:

```
Heuristic search completed
   Total number of rearrangements tried = 159267
   Score of best tree(s) found = 4939
   Number of trees retained = 1
   Time used = 1 sec (CPU time = 0.28 sec)
```

This shows that the length of the shortest tree is 4942. To find Bremer support values we need to retain more trees than just the most parsimonious tree. If the shortest tree in your search was 4942, to find which nodes have a decay index of 1, you need to find all the trees with 4943 steps or less. To do this we will use the keep option.

5. `hsearch keep=4943;`

The program may ask you to increase the maximum numbers of trees saved. Choose increase automatically. When the program is done, you'll see that there are more trees (29, if you are using anolis.nex). Use

6. `showtrees all;`

to see all the trees. Now, construct a strict consensus tree (saving the file is optional)

7. `contree all /treefile=`*filename*`.tre`

All the clades that are now unresolved do not appear in one of the trees that was one step longer, so they have a decay index of 1. To find higher Bremer support values increase the number of steps in the minimum trees:

8. `hsearch keep=4944;`
9. `contree all /treefile=`*filename*`.tre`

Until the tree is totally unresolved. Rather than doing a new search each time, you can reverse this process by doing just one search for trees that are a lot longer (say 5 steps):

10. `hsearch keep=4947;`

which produces 220 trees. Look at the consensus and note which nodes are preserved:

11. `contree all /treefile=`*filename*`.tre`

Then use the filter command to look at each set of better trees:

12. `filter maxscore=4946;`
13. `contree all /treefile=`*filename*`.tre`
14. `filter maxscore=4945;`
15. `contree all /treefile=`*filename*`.tre` …etc.