Lean on me: Finding Support for your Trees using Bootstrap, Jackknife, and Bremer Support

So ran a heuristic search and you got a tree. Now what? How do you tell how well (or poorly) supported the tree you've come up with is? Well, of course the truth is that for most cases in phylogenetics, it is impossible to know how closely your tree matches evolutionary history. Nonetheless, there are various different ways to get a sense of how robust your data is – that is, is that final 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 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, and is sometimes used only in bootstraps for this reason) or likelihood (which may take a long time).

Build a test tree

If you'd like to use your own data, please do. Otherwise, you can use the primate data that we used last time for MrBayes. It's in the MrBayes file in program files and also available on the syllabus page of the class website.

1. Place the file you'd like to work with in a new folder on the desktop.
2. Execute the file in PAUP.
3. `set criterion=parsimony;`
4. `hs;`

If you got the same results as me, the search found two trees. You can view a consensus tree using

5. `contree 1-2;`

or

`contree all;`

If you want to save the consensus tree to a file, use

6. `contree all /treefile=`*filename*`.tre;`

Estimating support by bootstrapping

Ok, 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 choose columns randomly from the matrix – until it has chosen the same number of columns as were in the originally 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. 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. To run a bootstrap analysis with a hundred replicates, using a heuristic search for each replicate, and randomizing the order of taxa while building trees, on your data, type

7. `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 too long at all. When it's done, PAUP will display a 50% majority rule consensus tree from all 100 bootstrap runs. Does the bootstrap support all the nodes that appeared in the Maximum Parsimony tree? At which nodes does it differ? It 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 (4) after it and Gorilla has a (5). Look down the chart until you see a line where there is a * under 4 and 5, but not under any of the other numbers. The frequency on this line is 54.92, which means these two taxa were grouped together in about 54 out of the 100 runs. (It's 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:

8. `savetrees from=1 to=1 savebootp=nodelabels file=`*filename*`.tre;`

Not 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 in Mesquite or Treeview. It is also supposed to work in TreeviewX, but I couldn't get it to. You may want to download Treeview to the desktop (http://taxonomy.zoology.gla.ac.uk/rod/treeview.html)

Estimating 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 a 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.

9. `jackknife nreps=100 search=heuristic /addseq=random;`

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

10. `savetrees from=1 to=1 savebootp=nodelabels file=`*filename*`.tre;`

Measuring node retention using 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 so tricks to get these numbers. Below, I will show you how to generate Bremer support numbers directly at the command line. 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 the program TreeRot.

You may remember finding Bremer support from the lab on "Advanced PAUP," but I'll go over it again anyway. First, you'll need to switch to a new data file—the primates.nex file isn't a very good example for Bremer support because everything has a really high decay index.

11. 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.
12. Execute the file in PAUP
13. `set criterion=parsimony;`
14. `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 = 35292
   Score of best tree(s) found = 4942
   Number of trees retained = 1
   Time used = 0.14 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.

15. `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 (2, if you are using anolis.nex). Use

16. `showtrees all;`

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

17. `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:

18. `hsearch keep=4944;`
19. `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):

20. `hsearch keep=4947;`

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

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

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

22. `filter maxscore=4946;`
23. `contree all /treefile=`*filename*`.tre`
24. `filter maxscore=4945;`
25. `contree all /treefile=`*filename*`.tre` …etc.

ILD (incongruence length difference)

I didn't have time to prepare an example for this, mostly because it requires a partitioned data set (ie, one with more than one source of data, like two different genes, or genes and morphology.) But, I wanted to explain how to do it anyway:

Compare the length of the most parsimonious trees for two or more data matrices or partitions to their length in the combined analysis and/or to randomly sampled partitions of equal size.

ILD = LAB -(LA + LB)/LAB

Significant incongruence suggests that partitions may have a different evolutionary history.

Some people would not combine data that showed significant incongruence. However, without evidence that there is some process that would cause the incongruence, down-weighting or eliminating character data simply because it is incongruent is really not scientific. Others would combine the data but consider that the result heuristically points to a need for more study. Examining RI and CI (see below) of partitions would be as informative and would allow for all data partitions to be examined in light of all critical evidence.