UPRM Biology
Syst Biol Lab 9

Name: _____ Date: _____

Lab 9 – Assessing Clade Support: Bremer Support, Bootstrap and Jackknife

Part I – Bremer Support: Introduction

In this lab we will ***assess the support that a character matrix lends to particular clades*** via the method of (1) Bremer support. ***Bremer support,*** a term coined by Kallersjö *et al.* (1992), is sometimes also referred to as the "decay index" (Donoghue *et al.,* 1992), or simply as "branch support" (Bremer, 1994). This measure is commonly used – and particularly in morphological analyses – to evaluate the support for clades. As noted by Davis (1993), the quantity obtained with Bremer support is *not* an index. Rather, it is a *raw value,* where ***higher scores*** are interpreted as ***higher support*** rather than higher decay. Therefore, the use of the term "decay index" for this measurement is somewhat misleading. Also, to be considered an index the value obtained must be derived from a mathematical formula.

This method was originally proposed by Bremer (1988). Bremer support for an individual clade, as evaluated by this method, is quantified as the ***difference in length*** between (1) the most parsimonious trees in which a clade *is* resolved and (2) the shortest trees – which are by definition longer than the most parsimonious trees – in which the clade is *not* resolved. Therefore, any clade resolved in all most parsimonious trees will have a Bremer support of at least 1.

Exercise 1: Calculating Bremer support

We will perform Bremer support assessment using NONA; in four different ways…

1. Using WinClada, open the character matrix ***lab9.ss.*** The matrix corresponds to an analysis of 32 plant genera. Run NONA (via the "Spawn" option) on the character matrix, and identify the most parsimonious trees (= "MPTs") using ***hold 2001; hold/20; mult*100;*** Save the resulting MPTs via the command ***sv* lab9.tre sv/;***

2. Set the default option to print trees in "ASCII" format so that you can view the resulting trees in any word processor. To do this, type ***txascii-;*** in NONA.

3. Set an output file to save the results of Methods 1-4 described below. ***Keep the output file open*** while doing all the following methods. If you follow the directions, it will remain open as long as is necessary. Type ***out lab9.out;*** All your results will be saved in this output file which you will examine after performing the four different methods.

4. Save the strict consensus tree with the command ***int lab9.nel sv/;***

Method 1

Use the "bs" command in NONA to calculate the Bremer support for different suboptimal tree ranges. The ***bs N;*** command calculates Bremer support values for the clades that occur in the shortest tree(s) in memory by performing TBR swapping on available trees – i.e., in this case, the four most parsimonious trees found above. The program then keeps X trees – in this case, up to 100 trees – that are up to N steps longer than the shortest tree(s) in memory, where N is the maximum number of additional steps

allowed in the suboptimal trees. For Method 1, the support values will be mapped onto the clades present in the consensus ***lab9.nel.***

a. Set to 100 the maximum number of trees to be held: ***hold 100;***
b. Type ***bs 10;*** – to keep trees up to ten steps longer than optimal in memory;
c. Type ***bs 4;*** – to keep trees up to four steps longer than optimal in memory;
d. Type ***bs 2;*** – to keep trees up to two steps longer than optimal in memory.

Method 2

Search for your own suboptimal trees and then perform "bs" on these trees.

a. Remove trees identified using Method 1 from the RAM by typing ***k0;***
b. Set NONA to keep trees up to ten steps longer than optimal by typing ***suboptimal 10;***
c. Use the "find*" command to perform TBR branch swapping: ***find*;***
d. Now, run "bs;" on the trees you found by typing ***bs 10;*** then ***bs 4;*** and finally ***bs 2;***

Method 3

Use the "mult" option to search for suboptimal trees.

a. Remove trees identified using Method 2 from the RAM by typing ***k0;***
b. Set the suboptimal tree search with "mult" to 10 by typing ***mult/10;***
c. Perform 50 TBR branch swapping replicates on the obtained trees by typing ***mult*50;***
d. Now, run "bs;" on the trees you found by typing ***bs 10;*** then ***bs 4;*** and finally ***bs 2;***

Method 4

a. Leave NONA open, and return to WinClada.
b. Run a ***bootstrap analysis*** on the character matrix; proceed to "Boostrap/Jacknife/CR with NONA" under the "Analyze" menu; use the following parameters: ***100 replications, mult*10, and hold/2;*** then enter ***lab9*** as the "stem name", and select the option to ***save ALL trees*** (i.e., not just the strict consensus). Hit the "Bootstrap" button. A new NONA window will open to run this analysis and will eventually close when it is complete.
c. Return to the original NONA screen, i.e. the one in which you are performing the Bremer support.
d. Remove trees identified using Method 3 from the RAM by typing ***k0;***
e. Reset the maximum number of trees to hold to 2000 by typing ***hold 2000;***
f. Read in the WinClada file that has the bootstrap trees: ***proc lab9.boo;***
g. Reset the memory to hold only 100 trees by typing ***hold 100;***
h. Now, run "bs;" on these bootstrap trees by typing ***bs 10;*** then ***bs 4;*** and finally ***bs 2;***

---

5. Close the out file by typing ***out/;***
6. Exit NONA with ***zz***

7. Now ***view the output file using WordPad*** (typically, this program is under "Start", "All Programs", and "Accessories" in the Windows environment. Open WordPad and proceed to the directory that you are using for the lab. Open the output file ***lab9.out*** and observe the results obtained from the different methods. ***Fill in the table below*** using the reference tree (i.e., the strict consensus ***lab9.nel;*** shown on this page) in order to compare the support for the corresponding clades. If a particular clade is not shown in the results for a given method, place an "X" in the cell for that clade.

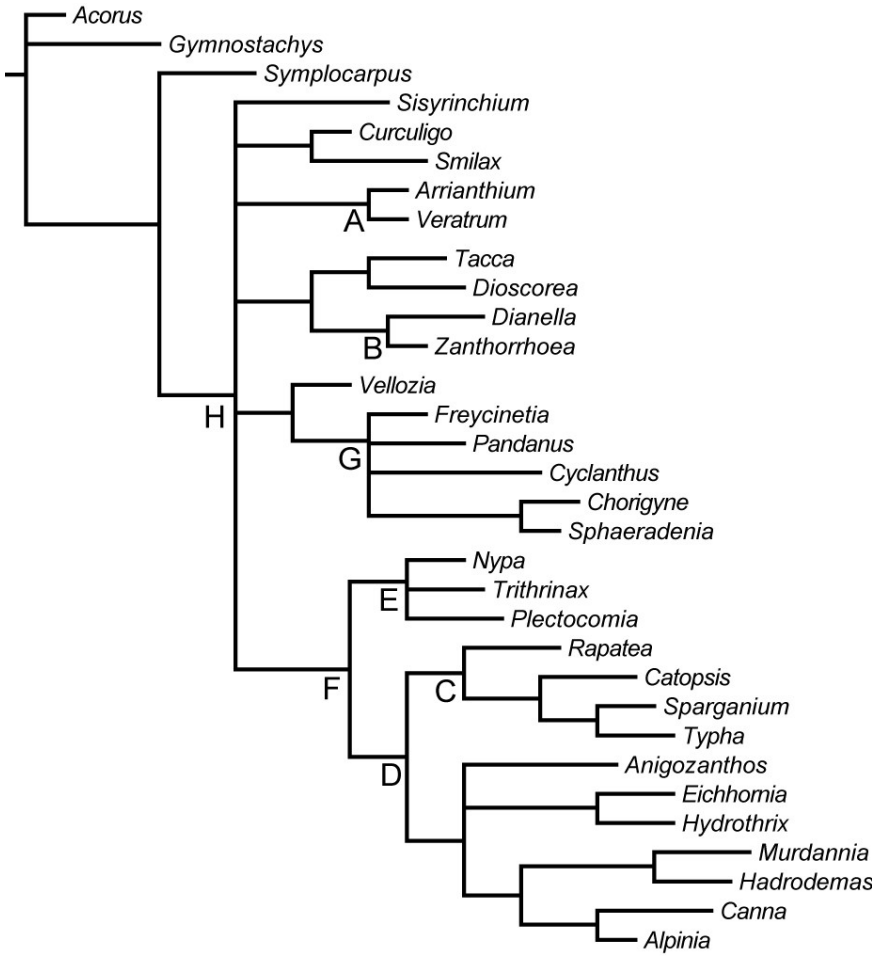


Table 1. Summary of Bremer support analyses.

	Bremer Method 1			Bremer Method 2			Bremer Method 3			Bremer Method 4			MPTs clade collapse			
	10	4	2	10	4	2	10	4	2	10	4	2	1	2	3	4
Clade																
A																
B																
C																
D																
E																
F																
G																
H																

Exercise 2: Calculating Bremer support by collapsing individual nodes (MPT clade collapse)

Bremer support can also be estimated by collapsing individual nodes.

1. Delete the bootstrap trees from WinClada via "DELETE all trees" under the "Trees" menu. Now read the shortest trees for the character matrix – ***lab9.tre*** – into WinClada. You should have only four most parsimonious trees with a length of 118 steps.

2. Set the mouse to ***collapse nodes*** by choosing "Collapse node mode" under the "Edit/Mouse" menu.

3. Record the length of the tree *after* you collapse each of the nodes of interest (see the reference tree on page 3 and the tree numbers on the table). ***Collapse one node at a time:***

- **a.** On the first MPT, collapse clade A.
- **b.** Check the length of the tree and subtract the difference.
- **c.** Fill in the table with this value.
- **d.** Reload the original tree by using the "Go to tree" option under the "Tree" menu, or use the "«" or "»" buttons for the same purpose.
- **e.** ***Repeat the procedure*** for each of the ***8 clades*** on each of the ***4 MPTs.***

Questions

1. Did you obtain different values *within* each of the different Bremer support calculation methods; e.g., for Method 1 holding trees up to 10 steps longer vs. holding trees up to 4 or 2 steps longer? Why or why not? Hint: you should cite two reasons why or why not.

2. Did you obtain different values *between* the different Bremer support calculation methods (excluding for now the clade collapse method)? Why or why not?

3. How does clade collapse compare to the other Bremer support methods? Do you think that clade collapse is a better method than the other strategies you tried? Why or why not?

Part II – Bootstrap and Jackknife Support: Introduction

In this lab we will examine the ***bootstrap*** and ***jackknife*** methods used commonly ***as indices of clade support.*** We will use WinClada to generate bootstrap and jackknife values, as calculated with NONA.

Bootstrap

Bootstrap values, as developed by Felsenstein (1985), are obtained by ***randomly resampling characters, with character replacement*** after each sampling step, until a new matrix of the same size as the original matrix is created. In practice, the new matrix is created simply by reweighting the characters:

(1) all characters are given a weight of 0;
(2) a character is then randomly selected, and its weight is increased by 1;
(3) step 2 is repeated until the sum of weights in the matrix is equal to the original sum of weights.

Any particular character may be selected once in the sampling process, more than once, or never. Attention: the inclusion of characters that are ***not parsimony informative*** can affect bootstrap values. Such parsimony-uninformative characters will make no contribution to the overall tree topology, yet they count towards the sum weights (see [3] above). Consider what would happen if, through a chance sequence of sampling events, only uninformative characters were selected... In practice this is only a concern with small matrices, or when the number of uninformative characters is small relative to the number of informative characters (e.g., morphological matrices). In matrices with large numbers of uninformative characters relative to the number of informative characters – e.g., most molecular sequence matrices – the bootstrap values are not significantly biased.

The ***probability*** of a character being selected at least once during a bootstrap run is ***approximately 0.62.*** The resampled matrix is then analyzed as follows. The analysis may consist of anything from simply calculating a Wagner tree (viz. the "fast bootstrap" option in PAUP*), to using a simplified parsimony search (e.g., hold/1; mult*1; in NONA), to more complete searches (e.g., hold/20; mult*100;). Usually, ***the strict consensus tree from each replicate is computed,*** and then used to represent the results of that replicate in the rest of the calculations. Sometimes all obtained trees (not just the strict consensus) are sometimes used in subsequent calculations. However, this is not logically consistent since ***each replicate should be counted equally,*** instead of receiving a higher or lower weight in light of the number of trees resulting from the replicate analysis. In earlier versions of PAUP, all trees from all bootstrap replicates were used in subsequent calculations. Current versions of PAUP* still use all trees, yet each tree is weighted such that each replicate ultimately receives equal weight.

The process is of resampling to construct a new matrix is repeated several times – usually 100 to 1000 replicates are performed. The results are obtained in the form of ***percentage values*** indicating the ***frequency*** with which each clade occurs among all of the replicates. The exact meaning of a particular bootstrap value is not straightforward (!), although bootstrap values are interpreted to mean anything from totally meaningless, to (most reasonably) an estimate of ***statistical confidence in particular clades in relation to the matrix that was sampled,*** to (in some cases) a measure of "truth".

The percentage of clade occurrence in the bootstrap analysis is usually mapped onto the strict consensus tree. In some cases, the majority rules compromise tree (50% or higher) is calculated from all the bootstrap replicates, and presented in place of the consensus tree, with mapped bootstrap values. The latter is sometimes called the "bootstrap tree", though many authors will not clearly state which of the two possible presentation types was used.

Jackknife

The jackknife was presented by Farris *et al.* (1996) as a faster and possibly more adequate alternative to the bootstrap. The authors intended the values resulting from a jackknife analysis to come out almost

identical with those obtained from a correctly implemented bootstrap analysis. Therefore the probability of a character being retained in the resampled matrix is set to 0.62 – the same as for the bootstrap. As a result, a resampled jackknife matrix will contain approximately two thirds of the original characters (this number varies among replicates). This probability could be adjusted to any number (e.g., 0.85), but this is rarely done.

The major methodological difference between the jackknife and the bootstrap is this: ***the jackknife samples without subsequent character replacement,*** whereas the bootstrap samples *with* such replacement. Functionally, this difference means that ***characters*** that are included in the resampled (new) jackknife matrix ***all have the same weight,*** whereas those characters retained in the resampled bootstrap matrix have different weights (usually weights of 1-3, and potentially more). In addition, the ***jackknife is not sensitive to uninformative characters*** in the matrix.

A jackknife analysis should in principle be faster than a bootstrap analysis since there are fewer characters to optimize. However, in NONA this difference in processing speed is negligible. Perhaps this is due to the bootstrap's potential for reducing the degree of character conflict by giving some characters more weight. The implementation of the jackknife is ultimately faster because it limits itself to starting with Wagner tree and performing SPR swapping only. Note: when using PAUP* to carry out the jackknife, one must specify both the deletion frequency (37%) and check the "emulate Jac sampling" option.

Exercise 1: "Jockstrap"

1. Open ***lab10.ss*** in "Jock" – this is a older version of WinClada that has a command option that we will need for this exercise.

2. Select "Show char statistics" from the "View" menu, and then open the "command Dialog" from the same menu.

3. Now type the word "jock" in the command dialog and press enter. This will do the following: (1) all characters are given a weight of 0; (2) a character is then randomly selected and its weight is increased by 1; (3) step 2 is repeated until the sum of weights in the matrix is equal to the original sum of weights; (4) all characters with a weight of 0 are selected. This should sound something like the procedure for resampling a matrix for the bootstrap.

4. Working with a partner, select a character, any character, to evaluate. Each partner must use the same character.

5. Each member of the pair will repeat the "jock" command 50 times. In the end, you can therefore pool your results, for a total of 100 replicates. Note: there is no need to retype the command, simply click on "Execute Command", or press enter. ***For each jock replicate, record the following data*** (in the conveniently provided table):

a. The number of characters with a weight of 0. This is indicated as the number of selected characters (= the number in parentheses and highlighted green after "NCHAR=91").

b. The total weight of the "jocked" matrix (= the first number after "WTS").

c. Record the weight of your selected character.

d. Record the minimum and maximum character weight observed in the matrix.

Rep #	a	b	c	d
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				
21				
22				
23				
24				
25				

Rep #	a	b	c	d
26				
27				
28				
29				
30				
31				
32				
33				
34				
35				
36				
37				
38				
39				
40				
41				
42				
43				
44				
45				
46				
47				
48				
49				
50				

Questions

1. How many times was the character you selected included in the matrix?

2. How did the total weight vary from replicate to replicate?

3. What was the average number of unique characters included (those with positive weights) in a replicate? Show your calculations.

4. What was the highest weight achieved by any character in an iteration?

Exercise 2: WinClada's Bootstrap and Jackknife

Using the "jock" command, you could have analyzed each weighted matrix, then saved the consensus trees, and calculate the bootstrap values "by hand". Realistically, let's the automatic features in WinClada instead.

1. Open ***lab10.ss*** in WinClada (now using the new version of WinClada) and analyze the matrix with NONA. You should obtain 4 most parsimonious trees with a length L = 118. Calculate and save the consensus tree with ***int lab10.nel sv/;*** Close NONA.

2. Select "Bootstrap/Jackknife/CR with NONA" under the "Analyze" menu. Enter the following values in the dialog: ***100 replicates; mult*20; hold/20; random seed = 0; your_initals_h20m20.*** Be sure that ***"Don't do max*"*** and ***"Save Consensus"*** are checked. Then select ***"bootstrap".*** The results will automatically be returned to WinClada. Make sure that there are exactly 101 trees in memory (= the 100 consensus trees you generated, plus the majority rules consensus of those trees).

3. Now ***delete tree 101*** (the majority rule tree presently in display) with "Delete selected trees" under the "Trees" menu. Then go to "Nodes" and "Calculate frequencies of nodes"; confirm for "All trees" (there are 100 trees) and "Percentage frequencies. This command will compare all nodes of all 100 trees and compute their percent frequency.

4. Open ***lab10.nel.*** The numbers displayed at the nodes are the bootstrap frequencies mapped onto the previously obtained strict consensus tree (= tree 101).

5. Record the frequency of selected nodes (see attached tree) in the appropriate table.

6. Select "DELETE all trees" under the "Trees" menu.

7. ***Repeat*** steps 2-6 ***twice,*** yet ***change*** the hold and search replicate values to: ***mult*5; hold/5;*** and ***mult*1; hold/1;*** respectively Also change the stem name of the file to match the appropriate values.

8. Repeat steps 2-7, yet click on "jackknife" instead of "bootstrap" in step 2.

9. Send via e-mail all of the *.boo and *.jac files that you have created to mailto:nico.franz@upr.edu This way the course results can be compiled. In addition, also send the information you recorded in the tables.

10. Now add 40 uninformative characters to the matrix; simply expand the matrix ("Matrix" menu) and leave the 40 new characters as "inapplicable" (–). Repeat the ***mult*1; hold/1;*** bootstrap and jackknife analyses, as described above. Record the frequency of selected nodes (see attached tree) in the appropriate table.

Questions

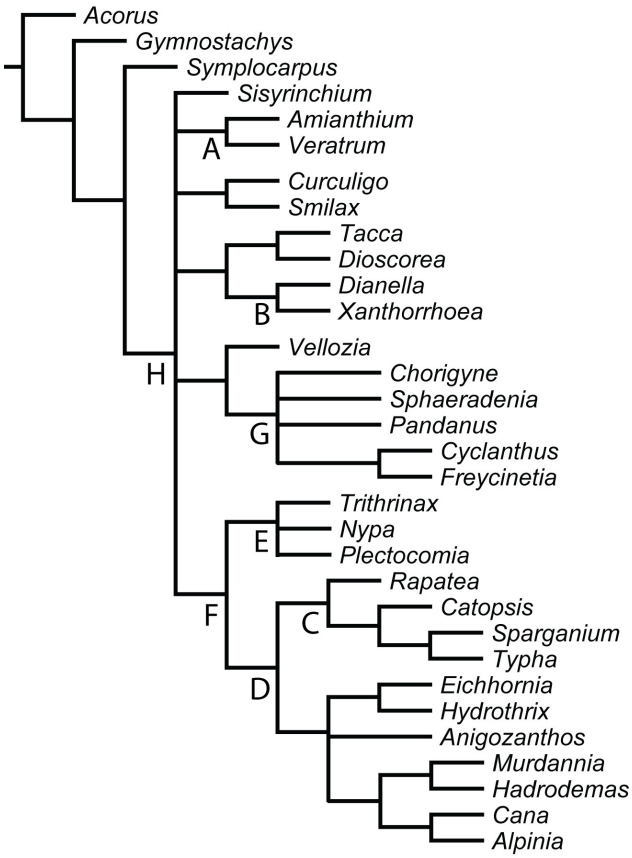
5. How do the bootstrap support numbers vary among the 100 replicate analyses? How different are the numbers in the pooled analysis in reference to your 100 replicate analysis?

6. Compare the results of the bootstrap ***hold/20; mult*20; search*** to those of the ***hold/1; mult*1; search.*** Can you suggest a reason why different bootstrapping strategies might result in different values for the same number of replicates?

7. Compare the bootstrap and jackknife analyses for the matrix with 40 uninformative characters.

8. How do the jackknife support numbers *vary* among the 100 replicate analyses? How different are the numbers in the pooled analysis, in reference to your 100 replicate analysis?

9. Compare the results of the ***jackknife hold/20; mult*20; search*** to those of the ***hold/1; mult*1; search.*** Can you suggest a reason why different jackknifing strategies might result in different values for the same number of replicates?



Bootstrap hold/20; mult*20;

Clade	Your result	Course min	Course Max	Pooled
A				
B				
C				
D				
E				
F				
G				
H				

Jackknife hold/20; mult*20;

Clade	Your result	Course min	Course max	Pooled
A				
B				
C				
D				
E				
F				
G				
H				

Bootstrap hold/5; mult*5;

Clade	Your result	Course min	Course Max	Pooled
A				
B				
C				
D				
E				
F				
G				
H				

Jackknife hold/5; mult*5;

Clade	Your result	Course min	Course max	Pooled
A				
B				
C				
D				
E				
F				
G				
H				

Bootstrap hold/1; mult*1;

Clade	Your result	Course min	Course Max	Pooled
A				
B				
C				
D				
E				
F				
G				
H				

Jackknife hold/1; mult*1;

Clade	Your result	Course min	Course max	Pooled
A				
B				
C				
D				
E				
F				
G				
H				

Bootstrap + 40: hold/1; mult*1;

Clade	Your result
A	
B	
C	
D	
E	
F	
G	
H	

Jackknife + 40: hold/1; mult*1;

Clade	Your result
A	
B	
C	
D	
E	
F	
G	
H	

References

Bremer, K. 1988. The limits of amino acid sequence data in angiosperm phylogenetic reconstruction. Evolution 42: 795-803.
Bremer, K. 1994. Branch support and tree stability. Cladistics 10: 295-304.
Davis, J. I. 1993. Character removal as a means for assessing stability of clades. Cladistics 9: 201-210.
Donoghue, M. J., R. G. Olmstead, J. F. Smith & J. D. Palmer. 1992. Phylogenetic relationships of Dipsacales based on rbcL sequences. Annals of the Missouri Botanical Garden 79: 333-345.
Farris, J. S., V. A. Albert, M. Kallersjö, D. Lipscomb & A. G. Kluge. 1996. Parsimony jackknifing outperforms neighbor-joining. Cladistics 12: 99-124.
Felsenstein, J. 1985. Confidence limits on phylogenies: an approach using the bootstrap. Evolution 39: 783-791.
Kallersjö, M., J. S. Farris, A. G. Kluge & C. Bult. 1992. Skewness and permutation. Cladistics 8: 275-287.

Homework Assignment (due in two weeks from today)

Complete all Exercises and answer all Questions presented in this Lab hand-out.