

# **BSC3402 (6): Experimental Biology — Comparative Genomics**

**Laboratory Section: Thursdays from 4:00 to 6:00 PM.**

## **Phylogenetic Analysis 1**

**Week Six, Thursday, February 15, 2007**

**Author and Instructor: Fredrik Ronquist**

Lab covers: (1) Running a program in the UNIX environment; (2) Introduction to PAUP; (3) An overview of the NEXUS format; (4) Neighbor-joining analysis using PAUP; and (5) Parsimony analysis using PAUP.

Fredrik Ronquist  
School of Computational Science  
Florida State University  
Tallahassee, FL 32306-4120  
ronquist@scs.fsu.edu  
850-645-1325

## 1. Introduction

In this lab, we will learn how to take a set of aligned, putatively homologous sequences and infer a phylogenetic tree for them using the methods of neighbor joining and parsimony as implemented in the software package PAUP (authored by Dr David Swofford of the Computational Evolutionary Biology group here at the School of Computational Science). In the process, we will learn how to run a program in the Unix environment (the Terminal window) on your classroom machines.

Rather than giving an elaborate introduction to phylogenetics, distance methods and parsimony here, I refer you to chapters 1 and 2 in Hall: *Phylogenetic Trees Made Easy*. In chapter 1, Hall describes how to download sequences from NCBI, align them in ClustalW, construct a neighbor-joining tree for them in ClustalW, and print the tree(s) using TreeView. ClustalW and TreeView are freeware and will run on all computer platforms; they also come on the CD included with Hall's book. Compared to using GCG, ClustalW and TreeView provide a simpler but much less powerful way of obtaining and analyzing a set of putatively homologous sequences. One of the few advantages is that ClustalW and TreeView do not require an X11 client. . Both GCG and ClustalW can save files in Nexus format, the format used by most phylogenetics programs.

On pages 61 to 102 of Chapter 2, Hall covers the basics of phylogenetic inference and describes neighbor-joining and parsimony analysis with PAUP, the topic of today's lab. A trial version of PAUP is also on the CD that comes with the book. In today's lab we will use the Unix version of the program, which is installed on your classroom machines. You may have noted that PAUP searches are also available as one of the many tools in the GCG package, so in principle we could have run PAUP from within GCG. However, running PAUP separately allows you to do much more with the program, and it is relatively likely that you will want to explore some of these extra options in your project, so we will run PAUP separately here.

## 2. Running a program in the Linux environment

Unlike the previous three labs, we will be running today's program, PAUP, on your own Linux machine instead of relying on the GCG package running on the Mendel server. To be able to do this, we need to access the command-line Unix system that runs underneath your windowing interface (GUI). Do this by clicking the **Terminal** symbol on your launch bar at the bottom of your screen. If you work outside of the classroom, you first need to connect to one of the classroom computers, for instance `class10.csit.fsu.edu`, using an SSH client program and your ordinary user name and password. Once the connection is established, your SSH client window should behave exactly the same way as the Terminal window on the machine you connected to, so you should be able to follow along in this tutorial without any problems.

When the Unix system is in control, it will print a prompt (usually terminated by `$`) and wait for your commands. The prompt is different depending on the flavor of your Unix system and your settings for the so-called *shell* (the terminal window). Typically, the system will indicate the directory (folder) you are working in. On my SSH terminal, the default prompt is

```
[ronquist@class10 ~]$
```

indicating that I am in my home directory (the symbol ~ is used as a synonym of the home directory) on the machine 'class10'.

When you type in a line and hit enter, the Unix system tries to find a program with the same name as the first word in your line. If it finds such a program, it will start that program and send the program the rest of the line you typed in. Typically, the program then interprets the rest of the line as instructions. All Unix commands you have used so far are of this type. For instance, when you type 'more test.txt', you start a program called 'more' and send it the line 'test.txt'. As you now know, 'more' will display the 'test.txt' file on screen, one screen at a time (if there is a 'test.txt' file in the directory you're in).

You have probably noticed that none of the Unix system programs, usually called commands, that you have used have resided in the directory you worked in. If you do 'ls' when you are in your home directory, for instance, you will not find the program 'ls' there. How can the Unix system find this program? The answer is that Unix uses your search 'path', which is a list of directories. It starts with the first directory in your path; if the program is not there it continues with the second directory, etc. If the program is in none of your directories, it will give you an error message. For instance, try typing the command **steve**, which should not correspond to any program in any directory in your path. Your system should produce the following output:

```
steve: Command not found.
```

Your path is contained in a so-called environment variable called \$PATH. To see your current path, type **echo \$PATH**. You should see something like

```
/usr/kerberos/bin:/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/usr/local/netbeans/bin:/usr/common/i686-linux/bin:/usr/local/condor/release-mnt/bin:/usr/X11R6/bin:/usr/local/netbeans/bin
```

Note that each directory is separated from other directories by colon signs. To be able to run paup, which we will be using for today's lab, you need to have the directory /usr/common/i686-linux/bin in your path. This directory will contain the program paup. To check that this is the case type **ls /usr/common/i686-linux/bin** and you should get a listing of all files in this directory, a long list of various programs. Note that the programs will be colored green to indicate that they are programs and not files (black) or directories (blue) [these colors appear only if you have the appropriate settings]. To run paup from anywhere on this computer, you only need to type paup and Unix will find the program and start it. To test that this works, type **paup**. You should see a welcome message as well as the paup prompt:

```
paup>
```

To leave the program, just type **quit** and you should get the Unix prompt back. How to change paths and set up links is outside of the scope of this tutorial but now you should at least be able to identify some simple

problems that might occur when you run Unix programs from a terminal window. If you cannot start paup, ask an instructor for help. You may want to refer back to the short list of Unix commands you prepared as part of the first lab. Many of these commands will be used today.

Before we start running the program, it is useful to set up a special directory containing the data and result files we will be working with. Create a directory called phylo1 in your home directory [**cd** to get to your home directory if you are not already there; **mkdir phylo1** to create the directory] and go there [**cd phylo1**]. Once you have created the directory you need to transfer the data files there (primates.nex and SRY.nex). They are available from the course site on Blackboard (see the Course Materials link).

## 2. Introduction to PAUP

Before running any analyses, it is useful to explore PAUP and some of the user help it provides. PAUP stands for Phylogenetic Analysis Using Parsimony (and Other Methods). PAUP can handle many different data types (amino acids, morphology, nucleotides, etc) and (despite its name) will do distance, parsimony, and maximum likelihood analyses. The program runs on all the common platforms but a complete GUI is only provided for Macs. If you bought the book by Barry Hall (Phylogenetic Trees Made Easy), you should have received a test version of PAUP. You can also buy the program from the publisher Sinauer or just use the Unix version provided for you this semester on your classroom machines.

Start the program again by typing **paup**. You should see the following information:

```
P A U P *
Portable version 4.0b10 for Unix
Mon Oct 11 19:37:51 2004
```

```
-----NOTICE-----
This is a beta-test version. Please report any crashes,
apparent calculation errors, or other anomalous results.
There are no restrictions on publication of results obtained
with this version, but you should check the WWW site
frequently for bug announcements and/or updated versions.
See the README file on the distribution media for details.
-----
```

paup>

If you type **help**, you will see a list of the available commands. It should look like this:

The following commands are always available:

!	Edit	Help	Quit
CD	Execute	Leave	Set
Defaults	Factory	Log	Time
DSet	FStatus	LSet	ToNEXUS

The following commands require data from a DATA (or TAXA and CHARACTERS) or DISTANCES block (\* = requires only TAXA block):

*Agree	*DerootTrees	*LoadConstr	Reweight	SurfCheck
AllTrees	*DescribeTrees	LScores	*RootTrees	*TaxPartition
AncStates	DScores	*MatrixRep	SaveAssum	*TaxSet
Assume	Exclude	MPRsets	SaveDist	*TreeDist

BandB	Export	NJ	*SaveTrees	*TreeInfo
BaseFreqs	ExSet	*Outgroup	ShowAnc	*TreeWts
Bootstrap	*Filter	PairDiff	*ShowConstr	*TStatus
CharPartition	GammaPlot	Permute	ShowDist	TypeSet
CharSet	*GenerateTrees	PScores	ShowMatrix	*Undelete
*ClearTrees	*GetTrees	PSet	ShowCharParts	UPGMA
Condense	HomPart	Puzzle	ShowRateSets	UserType
*Constraints	HSearch	RandTrees	ShowTaxParts	Weights
*ConTree	Include	*RateSet	*ShowTrees	Wts
CStatus	*Ingroup	Reconstruct	ShowUserTypes	WtSet
CType	Jackknife	*Restore	*SortTrees	
*Delete	Lake	RevFilter	StarDecomp	

Type "HELP COMMANDS" or "HELP CMDS" for a one-line description of each command.

Type "<cmdname> ?" to see brief usage and current default settings.

PAUP is a complex program and provides many commands, options and settings. The online help system only provides basic help on commands; a more complete manual is available at <http://paup.scs.fsu.edu/downl.html>. There are seven main commands that we will use today: 'Set', 'Dset', 'Pset', 'AllTrees', 'BandB', 'Hsearch', and 'ShowTrees'. The 'Set' command is used, among other things, to switch between different types of analyses. The 'DSet' and 'PSet' commands allow you to set the options for distance analysis and parsimony analysis, respectively. You typically adjust these settings before you start your analysis, or at least use these commands to go over the default settings to make sure you are happy with them. The 'NJ' command starts a Neighbor-Joining analysis. A parsimony analysis can be started using 'AllTrees' (an exhaustive search), 'BandB' (a Branch-andBound search) or 'HSearch' (a heuristic search). When an analysis is finished, you can see the best tree(s) found using 'ShowTrees'. We will need to use a few other commands as well, including 'execute' and 'outgroup'.

To see the available options and current settings for a command, type '<command> ?'. For instance, if you type **NJ ?** you get:

```
Usage: NJ [options...] ;
```

Available options:

Keyword	----	Option type	-----	Current default setting	--
Enforce		No Yes		No	
Constraints		<constraint-name>		<none>	
BioNJ		No Yes		No	
ShowTree		No Yes		Yes	
BreakTies		Systematic Random		Systematic	
TieSeed		<integer-value>		0	
TreeFile		<tree-file-name>		<none>	
Replace		No Yes		*No	
Append		No Yes		*No	
BrLens		No Yes		No	
				*Option is nonpersistent	

To take just one example, the setting of the option 'Showtrees' determines whether or not the resulting tree should be shown after an analysis. Before we can run any analyses with PAUP, however, we need to process a data file. For now, quit PAUP by typing **quit**.

### 3. An overview of the NEXUS format

PAUP and many other phylogenetic inference programs rely on input data being in a text file following the NEXUS format. We will be working first with the `primates.nex` datafile, which contains mitochondrial sequences from 12 different primates. To see the contents of the file, use **more** `primates.nex`. This is the structure you will see:

```
#NEXUS

begin data;
  dimensions ntax=12 nchar=898;
  format datatype=dna interleave=no gap=-;
  matrix
Tarsius_syrichta      AAGTTTCAT...
Lemur_catta          AAGCTTCAT...
Homo_sapiens         AAGCTTCAC...
...
Saimiri_sciureus     AAGCTTCAC...
;
end;
```

Note that the file starts with `#NEXUS`. This is required. The Nexus file is then divided into blocks, each of which begins with the statement `'begin <name of block>;'` and ends with `'end;'`. A file can contain many blocks; for instance, PAUP will read and understand a `'paup'` block that contains commands given in the same manner as if they were typed in from the keyboard, except that each command line needs to be terminated by a semicolon if it is contained in a `'paup'` block. This allows a user to start PAUP in batch mode, executing an analysis without user input. Hall covers PAUP blocks and running PAUP in batch mode on pp. 182-186.

Our file contains a single block, a `'data'` block. The block begins with a `'dimensions'` statement describing how many taxa (sequences) there are in the block (`'ntax'`) and how many characters (nucleotides) there are in each sequence (`'nchar'`). The next line specifies the format of the data. Here, the `'datatype'` is specified to be DNA, and format is not `'interleaved'` (that is, the entire sequence is on a single line), and `-` is used to denote gaps in the sequences (`'gap=-'`). Finally, the matrix is given; note the semicolon at the end of the matrix.

### 4. Running a Neighbor-Joining Analysis

Before we run any analyses, it is useful to check what is known about relationships among primates, so that we have some reference to evaluate our results against. Go to <http://tolweb.org> and see what information you can find there about primates and their relationships. Specifically identify the names and positions of the taxa in the `primates.nex` (which we will run first) and in the `SRY.nex` data sets (which we will run later).

Now it is time to produce some trees with PAUP. First, start PAUP by typing **paup** and then read in the data in the file `primates.nex` by typing **execute primates.nex**. Note that PAUP is not case-sensitive in general. PAUP is also able to decipher abbreviated commands if they are unambiguous. Thus, instead of typing “execute primates.nex” we could have typed “exe primates.nex”, PAUP would still understand what command to use. Note, however, that file names cannot be abbreviated.

PAUP should produce some lines of output to inform you that the data file has been processed. Now that the data are in the program, we can run a neighbor-joining analysis by simply typing **nj**. This starts the run using default settings for Neighbor-Joining. The program should return a single tree almost instantaneously. Does it appear familiar from the Tree of Life pages? Note that the tree has a basal trichotomy with *Tarsius syrichta* listed first. The basal trichotomy indicates that the tree is actually unrooted and has been rooted using *Tarsius syrichta* as the outgroup. The reason that *Tarsius* was chosen as the outgroup is simply that it was listed first in the data block. To change the outgroup to another species, for instance *Pan*, just type **outgroup pan**. Now run the Neighbor-Joining analysis again using **nj**. You should get the same tree except that it is now rooted at *Pan*. There is no way for PAUP to know where the tree should be rooted in this type of analysis; you have to provide that information based on data from other studies. Change the outgroup back by typing **outgroup tarsius\_syrichta**.

In some cases, ties occur during a NJ analysis and it is usually best to break them randomly even if this is not the default setting in PAUP. To see what option we need to change and how to change it, type **nj ?** to get:

Usage: NJ [options...] ;

Available options:

Keyword	----	Option type	-----	Current default setting	--
Enforce		No Yes		No	
Constraints		<constraint-name>		<none>	
BioNJ		No Yes		No	
ShowTree		No Yes		Yes	
BreakTies		Systematic Random		Systematic	
TieSeed		<integer-value>		0	
TreeFile		<tree-file-name>		<none>	
Replace		No Yes		*No	
Append		No Yes		*No	
BrLens		No Yes		No	
				*Option is nonpersistent	

To start a NJ analysis that breaks ties randomly, type **nj breakties=random**. Did this change the structure of the tree? Repeat the analysis a couple of times. Does the result change for each analysis? Is the result of this analysis sensitive to how ties are broken? Answer: This data set does not have any ties so the results are not sensitive to how ties are broken.

Finally, let us assess the phylogenetic uncertainty by using the bootstrap approach, invoking the ‘bootstrap’ command of PAUP. Before we do this, however, we must tell PAUP that we want to use the distance analysis criterion because PAUP needs to know that in order to interpret the bootstrap command correctly. By default, the criterion is set to parsimony, as you can see by typing **set ?** and examining the current setting for the

option 'criterion' (listed at the top of the option list). To change the search criterion to distance, type **set criterion=distance**. Type **bootstrap ?** to get an overview of the bootstrap options. Let us do 1,000 bootstrap NJ replications by typing **bootstrap search=nj nreps=1000**. You should very quickly get a bootstrap majority rule consensus tree back. What does this analysis tell you about the support for various groups in the NJ tree?

## 5. Running a Parsimony Analysis

Now let's build trees using the parsimony criterion instead. First change the criterion to parsimony using **set criterion=parsimony** (**set crit=pars** would also work because PAUP resolves abbreviated commands). Since we have a relatively small dataset, why not try the exhaustive search option first? Type **alltrees ?** if you are interested in seeing the available options. When you're satisfied, use **alltrees** (without the question mark) to start the analysis. PAUP will print its progress every minute; on my machine it took 3 minutes to finish about 11 % of the search, which means that the entire search would take about 30 minutes. This is a little bit too long for us so we stop the search by typing **<Ctrl> + C**. PAUP should not abort completely; if it does, just restart the program and read in the file `primates.nex` again. Normally, PAUP will ask you 'Do you really want to stop the search (Y/n)?'. Just type 'y' and hit return to stop the search.

Since exhaustive search was too time consuming, let us try Branch-and-Bound. First type **bandb ?** to see the available options. Note, for instance, that there is no initial upper bound on the trees to save ('UpBound = 0'), we will keep the shortest trees rather than all trees shorter than some length ('Keep = No') and we will keep all trees of minimum length ('MultiTrees = Yes'). This time, the default options are OK. When you are done examining the options, type **bandb** to start the search. After less than a second you should get the summary results back; there are two trees of length 1153. To see the trees we need to use the command 'showtrees' (if ever in doubt about what command to use, type 'help' to see a list of commands or go to the command reference at <http://paup.csit.fsu.edu/downl.html>). First type **showtrees ?** to see available options and then type **showtrees** to get the command started with the current settings. PAUP will now display the first tree of the two most parsimonious trees it kept from the analysis. To show both trees, you need to type **showtrees all**. The word 'all' is used instead of a list of tree names or tree numbers to indicate that you want to see all trees in memory; by default, showtrees only shows the first tree in the list. As an alternative, we could use 'showtrees 1-10', which would show trees 1 to 10 if there are that many trees in memory. This is the meaning of the '[tree-list]' item that you see in the usage statement when you type 'showtrees ?'.

Compare the two trees resulting from your parsimony analysis. How do they differ? What does this tell you about the relative support for different groups in the tree?

Finally, let us assess phylogenetic uncertainty in the parsimony context using bootstrapping. First type **bootstrap ?** to see the options and current settings. Note that the 'bootstrap' command uses heuristic search by default. Since our data set is so small, we can do an exact branch-and-bound search instead. Let us do

1,000 replicates of branch-and-bound by typing **bootstrap search=bandb nreps=1000**. This analysis took about 36 seconds on my machine. The resulting tree should look familiar but note the added uncertainty concerning some groups compared with the Neighbor-Joining bootstrap analysis.

After the bootstrap analysis, the original two most parsimonious trees are replaced by the bootstrap majority rule consensus tree. To get back the original trees, we must perform the parsimony analysis again. Do this by typing **bandb** and display the trees by typing **showtrees all**. Finally, we will examine the consistency and retention index of these trees using the command 'pscores'. First type **pscores ?** and note that the consistency index (CI) and the retention index (RI) are not printed by default. To change this behavior and obtain the total length, CI, and RI for both trees in memory, type **pscores all /ci=yes ri=yes**. Note that the slash is needed to separate the option settings from the tree list specification 'all'. Finally examine the CI and RI values for your trees. Do these indices indicate that the quality of the data is good or bad?

## 6. Analyze the SRY data

Now let us analyze the SRY data that we collected earlier in the course. This is a larger dataset (more sequences) so we will have to modify some analyses. For instance, NJ bootstrapping might have to use fewer replications and, in particular, the parsimony search will have to use the heuristic algorithm. If you wish, you can try to figure out how to run these analyses yourself by using the **help** command and the **<command> ?** way of getting help about the usage of a particular command. You may also want to look back at the analyses we ran with the primates.nex dataset. If you are less adventurous, you can continue reading for more detailed instructions.

Before you start the analysis of the SRY dataset, you need to read it with the **execute SRY.DNA.nex** command. Before we perform the neighbor joining analysis, let us check the current settings for this type of analysis by typing **nj ?** The output should look like this:

```
Usage: NJ [options...] ;
```

```
Available options:
```

```
Keyword ---- Option type ----- Current default setting --
Enforce      No|Yes                      No
Constraints  <constraint-name>           <none>
BioNJ        No|Yes                        No
ShowTree     No|Yes                        Yes
BreakTies    Systematic|Random            Random
TieSeed      <integer-value>              132206511
TreeFile     <tree-file-name>            <none>
Replace      No|Yes                        *No
Append       No|Yes                        *No
BrLens       No|Yes                        No
                                     *Option is nonpersistent
```

Remember that we changed the setting of 'Breakties' to 'Random' previously? Note that this setting still persists, it has not gone back to the default for PAUP, which is 'Systematic'. If you restart PAUP, the

setting will revert back to the default but as long as you run the program, most changes of parameter settings are persistent. A few are not, as noted in the options table.

To run your neighbor-joining analysis, just type **nj**. After having examined this tree, run a bootstrap analysis to look at the support for various groups in the tree. Before you do that, remember that you need to set the search criterion to distance by typing **set criterion=distance**. If you forget to change the search criterion and it is currently set to parsimony, PAUP will warn you when you try to do the neighbor-joining bootstrap analysis by telling you:

```
Error(#425): Neighbor-joining or UPGMA bootstrap is permitted only with distance
optimality criterion.
```

If you see this message, just change the criterion to distance as explained above.

Now run the bootstrap by typing **bootstrap search=nj nrep=100**. This analysis should complete so quickly that you might consider rerunning the analysis with 1,000 replications. Remember that you can roughly predict the time it will take to complete a certain number of bootstrap replications by simply multiplying the time for a single analysis with the number of replications. The time will only be approximate because each bootstrap replicate is special and the time taken to complete the analysis can vary quite considerably, especially for heuristic searches like the ones we will use below. If, by mistake, you start an analysis that takes too long, just stop it by pressing `<ctrl>+C` and then answering 'y' when PAUP asks you if you really want to stop the analysis.

Now switch to parsimony by typing **set criterion=parsimony**. When working with the parsimony criterion, we have to use the heuristic rather than any of the exact approaches to tree searching because we have so many taxa in the dataset (it is the number of taxa, not the length of the sequences, that largely determines the computational complexity of the search). The heuristic search command in PAUP is called **hsearch**. Type in **hsearch ?** to see the available options for the hsearch command. Note that the default setting is to perform stepwise addition (`addseq=simple`), holding 1 tree at each step (`hold=1`), followed by TBR (Tree Bisection and Reconnection) swapping (`swap=TBR`). These are reasonable settings for our heuristic search. However, there is a couple of overall settings that we might want to change first. For the overall program settings, type **set ?** This will produce the following, long list of program settings:

```
Usage: Set [options...] ;
```

```
Available options:
```

Keyword	----	Option type	-----	Current default setting	--
Criterion		Parsimony	Likelihood Distance	Parsimony	
RootMethod		Outgroup	Lundberg Midpoint	Outgroup	
OutRoot		Polytomy	Paraphyl Monophyl	Polytomy	
Monitor		No	Yes	Yes	

MaxTrees	<integer-value>	100
Increase	No Prompt Auto	Prompt
AutoInc	<integer-value>	100
InitSeeds	0 1	0
ShowExcluded	No Yes	No
CMLabels	No Yes	No
CMShowEq	No Yes	Yes
CMColWid	<integer-value>	1
CMCstatus	No Yes	No
Constraints	<constraint-name>	<none>
Status	No Yes	Yes
DStatus	<integer-value> None	60
CheckEvts	No Yes	Yes
AutoClose	No Yes	No
Background	No Yes	Yes
VisNotify	None ShowAlert FlashOnly	ShowAlert
NotifyBeep	No Yes	Yes
AllDigLab	Prohibit Warn NoWarn	Warn
AllowPunct	No Yes	No
DefaultMode	No Yes	No
ShowAbbrev	No Yes	No
SemiGraph	No Yes	No
TOrder	Standard Right Left Alphabet	Standard
TCompress	No Yes	No
ShowTaxNum	No Yes	No
TaxLabels	Truncate Full	Truncate
StoreBrLens	No Yes	No
StoreTreeWts	No Yes	No
FlushLog	No Yes	No
ErrorStop	No Yes	Yes
WarnReset	No Yes	Yes
WarnTree	No Yes	Yes
WarnTSave	No Yes	Yes
WarnBlkName	No Yes	Yes
WarnRoot	No Yes	Yes
WarnRedef	No Yes	Yes
ErrorBeep	No Yes	Yes
QueryBeep	No Yes	No
KeyBeep	No Yes	No
BrlenEpsil	<real-value>	1e-08
TabSpaces	<integer-value>	4
Pause	No Silent Beep Msg	No
Flock	No Yes	Yes

Note the fourth and fifth settings. They determine what happens if PAUP encounters multiple equally optimal trees during a tree search. By default, it will save only the first 100 trees (MaxTrees = 100) and it will then prompt you if you want to save more trees (Increase = Prompt). It is typically more convenient if we have PAUP automatically store all the equally parsimonious trees it finds during the search by automatically increasing the maximum number of trees. Do this by typing **set increase=auto**. Now we can run the heuristic search using the default settings by simply typing **hsearch**. The search took about 4 minutes on my machine and resulted in a large number of equally parsimonious trees. It doesn't make much sense displaying all trees using 'showtrees all' as described above. If you want to see one tree, you can type **showtrees 1** (to display the first tree in memory; actually, you do not even have to type '1', just 'showtrees' will do the same thing). A useful technique to summarize a large set of most-parsimonious trees is to compute a majority-rule consensus tree, just like the one computed automatically for a set of bootstrap replicates. To do

this, just type **contree**. If you give `contree ?` first, you will see that the default behavior of the command is to construct only a strict rule consensus tree. The strict-rule consensus tree is similar to a majority-rule consensus tree except that it only keeps those groups that are found in all of the trees that are summarized. If you are interested, you can also construct a majority-rule consensus tree by typing **contree / majrule=yes**.

If you wanted to check whether it is likely that your heuristic search found all of the shortest trees, you can try running a few random addition sequences. This will spawn a number of heuristic searches, each starting from a slightly different tree. For instance, run four random addition sequences by typing **hsearch addseq=random nreps=4**. If the analysis takes too long, interrupt it with **<ctrl>+C**. Finally, if we wanted to run a parsimony bootstrap analysis, we first need to figure out a way of making the heuristic searches less comprehensive so that the bootstrap analysis can finish in time. For instance, try a single analysis using NNI (Nearest Neighbor Interchanges) instead of TBR by typing **hsearch addseq=simple swap=nni**. This analysis should be sufficiently fast for us to use it as the basis of a bootstrap run. Do this by typing **bootstrap search=heuristic / addseq=simple swap=nni**.

When you have examined the results, think about their implication. What do the SRY data tell us about primate relationships? How do these results agree with conventional wisdom and with the small mitochondrial data set we just analyzed?

## 7. Homework Assignment

Use your own data and run a neighbor-joining and a parsimony analysis. Present the results from these analyses in the form of two trees, one from each type of analysis. At least one of the trees must have support values (bootstrap values) indicated. Also include one or two paragraphs describing your analyses and discussing the results.