Lab #2 Maximum Parsimony using PAUP and TNT

The objective of this lab is to learn how to do a parsimony analysis using PAUP and TNT.

PAUP* Tutorial

Ockham chooses a razor

PAUP* (pronounced "pop star") is a phylogenetic program first developed in 1993 by David L. Swofford, currently at Duke University. The name PAUP used to mean "Phyogenetic Analysis Using Parsimony" because parsimony was the only optimality criterion employed at the time. The asterisk in the name PAUP* means and other methods, which were added later. The new name is "Phyogenetic Analysis Using PAUP". PAUP* 4.0 is able to perform distance, parsimony and likelihood analyses using DNA and amino-acid data. Among many strengths of the program are a rich array of options for dealing with phylogenetic trees including importing, combining, comparing, constraining, rooting and testing hypotheses. The program has been updated recently and is expected to be released as an open-source command-line version (but don't hold your breath!). We will be using PAUP for this lab because of its transparency in dealing with the parsimony analysis. MEGA 7 is an alternative (and free for academic use) program that can perform maximum parsimony and several other analyses.

First things first

You have two options for running PAUP: you can run it on your computer or use the HPC-class server. I'll describe the steps for running it on HPC-class.

1. Open you terminal and ssh to HPC class: `ssh isu_name@hpc.itd.iastate.edu` ;
2. While you are in the home directory, modify your `.bash_profile` file:
    - open it in vi: `vi .bash_profile` ;
    - use down arrow to go to the `PATH=...` line;
    - type `A` and add the following text to the end of the line: `:/shared/class/eeob563`
    - press `ESC`
    - type `ZZ` to save and quit the file;
3. Log out of HPC-class and log in again.
4. Create an eeob563 directory if does not exist: `mkdir eeob563` and go to it;
5. Within the eeob563 directory clone the git repository for the class: `git clone https://github.com/ISU-MolPhyl/EEOB563-Spring2018` .
6. Also clone the "labs" repository you created as part of the assignment 2.
7. Witin the labs directory create a new directory called lab_2 and switch to it.

Getting the sequences

1. We will use a set of cytochorome b aa sequences for this exercise. The sequences are in EEOB563-Spring2018 repository. Copy them to your lab_2 directory: `cp ../../EEOB563-Spring2018/computer_labs/lab2` . If you still have problems with GitHub, you can download sequences directly into your directory using the following unix command: `wget https://sites.google.com/site/eeob563/computer-labs/lab-2/cob_aa.fasta`
2. Align these sequences using mafft and save them in fasta format.
3. To use this alignment in PAUP, we'll need to convert it into the Nexus format.
4. Read brief Wikipedia descriptions of FASTA and Nexus file formats.

Starting PAUP

1. Start PAUP* by typing:
   `paup`
2. If you will remember just one command in PAUP, remember this one:
   paup> `help` (note that I put "**paup>**" in front of the command to indicate that you are running it in PAUP!
3. Notice that there are many commands available in PAUP! We will only use a subset of them today. To get an idea for what they do, type
   paup> `help cmds`
4. Let's figure out how to import our file into Nexus format:
   paup> `ToN ?`
   Notice, that `<cmdname> ?` shows not only how to use a command, but also current/default settings!
5. Convert your aligned file into Nexus format;
6. Finally, we need to execute this new file (you'll see why later):
   paup> `exe cob_aa.nxs`
   Did it work? If not, try to see what went wrong by using the edit command (which opens the vi editor inside paup):
   paup> `edit cob_aa.nxs`
   After you executed the file, you should get a note that "Processing of input file "cob_aa.nxs" completed.

Congratulations! For additional information see PAUP 3.1 manual (obsolete, but has an excellent background section on parsimony analysis) and the command reference document.

Running maximum parsimony searches in PAUP

1. Before we run the analysis in PAUP, it is a good practice to start a log file. This log file will store all of the output that PAUP* presents in the Terminal window and will be helpful to keep a record of your analysis for future reference:
   paup> `log file=paup_mpsearch.log`
2. As we discussed in class, there are several strategies we can use to search for the most parsimonious tree. Let's start by trying to do an exhaustive search:
   paup> `AllTrees`
   Were you able to run this search?
3. Let's try to use branch and bound search instead:
   paup> `BandB`
   Were you able to run this search?
4. Let's remove some sequences. But first you need to cancel your ongoing search by pressing "control+c" (the same combination is used to cancel most of the unix commands). Now, let's look at the taxa in our dataset
   paup> `tstatus full` We can also check the status of all characters:
   paup> `cstatus full`
   Let's delete non-placental mammals:
   paup> `delete 1-5`
   Also delete cat, dog, and pig. *How many taxa are left?*
5. Now we may be able to run the exhaustive search, but let's try to run branch and bound search instead.
   How long did it take to run this search?
6. Now let's return deleted taxa in the dataset: **paup>** `undelete all`
7. We will run a heuristic search instead. Check the options:
   paup> `hsearch ?`
   Recall, we do heuristic search by making an initial tree, rearrange its branches, and repeat this process multiple times. Can you find options for these steps? A reasonable strategy for running a heuristic search would be to use use a random stepwise addition sequence with TBR branch swapping and 100 replicates:
   paup> `hsearch start=stepwise addseq=random nreps=100 swap=TBR`
   What was the best score found by this search? How many trees had this score? Do we get the same two trees if we increase the number of replicates to 1000?

8. There are several commands to look at the best trees:
   paup> `showtrees all`
   paup> `describe`
   We can define an outgroup before displaying the trees:
   paup> `outgroup 1-2`
   We can also incorporate branch length information by describing the optimal tree as a phylogram (a tree with parsimony branch lengths).
   paup> `describe /plot=phylo`
   Note the forward slash (/); this is important for many commands. Values before the slash indicate which trees you wish to include, options that apply to the primary command (describe) are placed after the slash;
9. We can also make a consensus tree to see how the trees differ from each other: **paup>** `contree`
10. Save the trees to file.
   paup> `savetree file=hsearch.tre brlen=yes`
11. We have now finished our heuristic search. So now, let's clear the trees from memory, stop our previous log file, and reset the defaults for the Heuristic search.
   paup> `cleartrees; log stop; defaults hsearch; quit;`
12. OK, we are out of PAUP now! Let's look at the logfile:
   `cat paup_mpsearch.log`

Conducting a Parsimony Bootstrap Analysis

1. Start PAUP*
   `paup`
2. Execute the example data file:
   paup> `exe cob_aa.nxs;`
3. Start a log file:
   paup> `log file=paup_bootstrap.log;`
4. Set the rooting:
   paup> `outgroup 1-2; set outroot=mono;`
5. To run a bootstrap search we need to specify details of the bootstrap search and the heuristic search of each bootstrap pseudoreplicate. The set-up should look like this:
   paup> `bootstrap [bootstrap options] [/hsearch options]`
   Here, the forward slash separates the bootstrap options from the heuristic search options for each bootstrap pseudoreplicate.
   paup> `bootstrap nreps=200 treefile=boot.tre search=heuristic /start=stepwise addseq=random nreps=10 swap=TBR;`
6. If you want to save a tree that has the bootstrap proportions on it, you need to save this tree immediately after the bootstrap analysis is complete by typing: **paup>** `savetrees file=bootMajRule.tree from=1 to=1 savebootp=nodelabels;`
   In this command, it is necessary to specify the tree you'd like to save using the `from/to` option, even if there is only one tree in memory. We use the `savebootp=nodelabels` option to save the bootstrap proportions as nodelabels. These bootstrap values can then be visualized in other programs (e.g., FigTree).
7. We might also want to summarize our bootstrap proportions after the analysis was completed. You might do this if you were combining tree files from multiple analyses run on different computers. So let's make a new consensus tree from our tree file. Open the treefile by typing:
   paup> `gettrees file=boot.tre StoreTreeWts=yes mode=3;`
   We use `StoreTreeWts=yes` option to downwait multiple trees found for some bootstrap pseudoreplicates. This is important for bootstrap and jackknife replicates and the majority-rule consensus calculator will use them when `USETREEWTS= YES`. We use `mode = 3` (DEFAULT) option to replace all of the trees currently in memory with those imported from file.
8. Now let's make a majority rule consensus tree and save this tree (see step 6). **paup>**
   `contree all/strict=no majrule=yes usetreewts=yes treefile=bootMajRule.tre;`
9. We have now finished our bootstrap analysis. Stop the log file and quit PAUP:
   paup> `log stop; quit;`

Using PAUP Blocks

In the previous two section, we used PAUP in an interactive mode. An alternative way to use the program is to put the necessary instructions in a special PAUP block in the original nexus file. These blocks are very useful for several reasons. First, your nexus file will serve as a permanent record of what you did. Second, such files can be submitted in the batch mode on remote clusters (such as HPC-class). Finally, using written commands will reduce the introduction of error that can occur when using the command line.

Let's modify the cob_aa.nxs file to include a PAUP block.

Open the cob_aa.nxs file in a text editor. Add two additional blocks under the data matrix:

```
BEGIN SETS;
CHARSET beginning = 1-12;
CHARSET end = 379-382;
TAXSET outgroups = 1-2;
END;

BEGIN PAUP;
exclude beginning end;
outgroup outgroups;
set increase=auto autoclose=yes;
hsearch start=stepwise addseq=random nreps=20 swap=TBR;
END;
```

The top block defines two character sets, one includes 12 characters and the other includes 4 characters, and a taxon set termed "outgroups" that includes two taxa.

The PAUP block uses `exclude` command to explute two character sets and `outgroup` command to treat two taxa as outgroups (but remember, that changing outgroups only changes the way the tree is displayed and does not change the parsimony score!) In addition, a heuristic search is specified.

Now execute this file in PAUP using the `execute` command:
paup> `exe cob_aa.nxs`
You can also specify the file at the same time as you invoke paup:
`paup cob_aa.nxs`

I attached some additional examples of PAUP* blocks for your reference. An excellent tutorial on PAUP that goes beyond parsimony can be found here.

TNT Tutorial (optional)

TNT (Tree analysis using New Technology)

A parsimony program by Pablo Goloboff, Steve Farris, and Kevin Nixon that can:
- analyse large data sets (i.e. 300-500 taxa);
- in reasonable times (minutes to find a shortest tree, hours to produce a reliable consensus).

Described in Goloboff PA, Farris JS, Nixon KC (2008). TNT, a free program for phylogenetic analysis. Cladistics 24:774–786.

The input data file can be in either TNT or basic Nexus format. The TNT format is derived from Hennig86 ⁄ NONA. The data file should start with a specification of the data type, which in the case of DNA/AA data is nstates dna/prot;. Next comes the xread command, followed by the number of characters, the number of taxa, and the data themselves (sequence data must be prealigned). Character states may be IUPAC codes, digits (for morphological characters), ? (for missing data), or - (for gaps).

Example:

> xread
> 1095 46
> Sequence*name #1*
> *Sequence #1*
> ...
> *Sequence*name #46
> Sequence #46
> ; ' Note the semicolon '

To read in the data file, type in `procedure [filename]` or just `p [filename]`, where filename is the name of your file. Usually you will name your data file with a .tnt extension, such as mydata.tnt. Before analyzing the data, you should make provision for saving the output to a log file. This can be done by entering `log [logfilename]`. Usually you should give a log file the .out extension and the same base name as the data file—something like mydata.out.

All program output is temporarily saved to an internal text-buffer. This text-buffer can be inspected with the `view` command.

A basic analysis consists of using multiple addition sequences followed by branch swapping. To do this enter `mult;`. By deafult, this command will perform 10 random addition sequences followed by branch-swapping, saving up to 10 trees per replication (roughly equivalent to a "heuristic search" with random addition of sequences in PAUP*). In the case of small data sets (15–30 taxa), exact solutions using branch-and-bound (which guarantee finding all trees optimal under current settings) can be produced within reasonable times with the `ienum` command.

Once calculated, trees may be viewed by entering tplot. To save the trees for later reanalysis, create a file by entering `tsave * [tree_filename]`. Usually you will name your save file with a .tre extension. If there are multiple trees, their strict consensus can be found by entering `nelsen`. Resampling (jackknifing, bootstrapping, etc.) can be done with `resample`. All available commands can be seen by typing `help` and information about each of them – by typing `help [command]`.

Now copy the file cox1_nt.tnt to your working directory and try this command:
`tnt p cox1_nt.tnt, echo=, log cox1_nt.out, rep+1, mu10=ho3, le, ne, resample, quit,`

What did you do and where are your results?

Additional information

TntWiki