SISG “Short” PAUP* Lab

Note: Parts of this computer lab exercise wer written by Paul O. Lewis. Paul has graciously allowed Mark Holder to use and modify the lab for the Summer Institute in Statistical Genetics. Thanks, Paul!

This computer lab you will introduce you to some basic aspects of PAUP*. Versions of PAUP* exist for several different operating systems (MacIntosh, Windows, Linux, etc.), with the MacIntosh version being the most flexible and user-friendly.

We are going to work from free (but expiring) versions available here: `http://people.sc.fsu.edu/~swofford/paup_test`

You can work through this tutorial at your own pace, asking questions whenever something needs to be clarified. Please let us know if you think another approach would be better, and if anything about this tutorial is unclear. The goals for this tutorial are to:

- Become familiar with the NEXUS data file format used by PAUP* (as well as several other prominent phylogeny programs such as Mesquite and MrBayes)
- Learn how to conduct various types of searches (exhaustive, branch-and-bound, heuristic using NNI and TBR branch swapping, and algorithmic approaches such as star decomposition and stepwise addition)
- Learn how to set up PAUP* to perform parsimony, minimum evolution, least squares searches (we will cover ML in the next lab).

Questions that you should be able to answer from looking at the output are in *italics*. Answers to the questions are provided in footnotes. If you do not understand one of these questions, or need help figuring out the answer, please do not hesitate to raise your hand.

Searching under the parsimony criterion

1. **Create the data file** Download the file angio35b.nex from http://www.people.ku.edu/∼mtholder/848/data/angio35b.nex and save it on the machine that you are working on and take a quick look at it. It contains a data block with the sequence matrix; A sets block that describes where the breaks between the different genes fall; and an assumptions block that tells PAUP* to exclude some characters that may not be aligned reliably.

2. **Create a command file.** Create a blank file, then type in the following commands, and save the file as run.nex in the same directory that holds the angio35b.nex file. Here are the PAUP* commands:

```
#NEXUS
begin paup;
    Log file=output.txt start replace;
    Execute angio35b.nex;
end;
```

3. **Execute run.nex, which will in turn execute angio35b.nex**. There at least two advantages to creating little NEXUS files like run.nex. For now, the only advantage is that executing run.nex automatically starts a log file so that you will have a record of what you did. Later, when you get in the

habit of putting commands in paup blocks, you will appreciate the separation of the data from the commands that initiate analyses (I have many times opened a data file, forgetting about the embedded paup block that then starts a long search, overwrites my previous log file, and otherwise creates havoc).

Note that because we used the `replace` keyword in the `log` command, the file `output.txt` will be overwritten without warning if it exists. This is a bit dangerous, so you may want to refrain from using the replace keyword so that PAUP* asks before overwriting files.

4. **Delete all taxa except the first five**. The command `delete 6 - .` will cause PAUP* to ignore all taxa except Ephedrasinica, Gnetum_gnemJS, WelwitschiaJS, Ginkgo_biloba, and Pinus_ellCH. Type that command into the command line of PAUP*. Note the `.` is part of the command – it stands for 'the last member in the list' (in this context it is 'the last taxon in the data matrix'.

5. **Perform an exhaustive search using parsimony**. Use the `alltrees fd=barchart` command for this. This should go fast because you now have only 5 taxa.

   - *How many separate tree topologies did PAUP* examine?*[1]
   - *What is the parsimony treelength of the best tree? The worst tree?*[2]
   - *How many steps separate the best tree from the next best?*[3]

6. **Perform an heuristic search using NNI branch swapping**. Before you start, use the `describe` command to show you the tree obtained from the exhaustive enumeration. Draw this tree on a piece of paper and then draw the 4 possible NNI rearrangements

   Find all NNI rearrangements of the best tree. Note that because we performed an exhaustive enumeration, we now know which tree is the globally most parsimonious tree. We are thus guaranteed to never find a better tree were we to start an heuristic search with this tree. Let's do an experiment: perform an NNI heuristic search, starting with the best tree, and have PAUP* save all the trees it encounters in this search. In the end, PAUP* will have in memory 5 trees: the starting tree and the 4 trees corresponding to all possible NNI rearrangements of that starting tree:
   ```
   hsearch start=1 swap=nni nbest=15
   ```

   - `start = 1` starts the search from the tree currently in memory (i.e., the best tree resulting from your exhaustive search using the parsimony criterion)
   - `swap = nni` causes the Nearest-Neighbor Interchange (NNI) method to be used for branch swapping
   - `nbest = 15` saves the 15 best trees found during the search. Thus, were PAUP* to examine every possible tree, we would end up saving all of them in memory. The reason this command is needed is that PAUP* ordinarily does not save trees that are worse than the best one it has seen thus far. Here, we are interested in seeing the trees that are examined during the course of the search, even if they are not as good as the starting tree.

   Show all 5 trees in memory. Use the `describe all` command to plot the 5 trees currently in memory. The reason we are using the `describe` command rather than the `showtrees` command is because we want PAUP* to show us the numbers it has assigned to the internal nodes, something that `showtrees` doesn't do.

[1] 15 topologies
[2] 1110 was the best score, and 1247 was the worst
[3] 13 steps – this is hard to see in the versions of PAUP posted on June/2011.

- *Which tree was the original tree?*[4]
- *Which trees correspond to NNI rearrangments of which internal edges on the original tree?*[5]

7. **Find the most parsimonious tree for all 35 taxa**. Restore all taxa using the `restore all` command (this will wipe out the 5 trees you currently have stored in memory, but that is ok), then conduct a heuristic search having the following characteristics:

    - The starting trees are each generated by the stepwise addition method, using random addition of sequences
    - Swap using NNI branch swapping
    - Reset the `nbest` option to `all` because we want to be saving just the best trees, not suboptimal trees (yes, this option is a little confusing).
    - Set the random number seed to 5555 (this determines the sequence of pseudorandom numbers used for the random additions; ordinarily you would not need to set the random number seed, but we will do this here to ensure that we all get the same results)
    - Do 500 replicate searches; each replicate represents an independent search starting from a different random-addition tree

    Here is the full command implementing this search:

    ```
    hsearch start=stepwise addseq=random swap=nni nbest=all rseed=5555 nreps=500
    ```

    - *How many tree islands were found?*[6]
    - *How long did the search take?*[7]
    - *How many rearrangements were tried?*[8]

8. **Conduct a second search with SPR swapping**. Be sure to reset the random number seed to 5555. You should be able to figure out how to do this using the output from `hsearch ?` command. Note that to save typing you can call up previously entered commands using the little buttons on the right of the command line edit control (or using the arrow up key).

    - *How many tree islands were found?*[9]
    - *What are the scores of the trees in each island?*[10]
    - *How long did the search take?*[11]
    - *How many rearrangements were tried?*[12]

9. **Now conduct a third search with TBR swapping.**

    - *How many tree islands were found?*[13]

[4]It should be the first one – the tree with score 1110.
[5]It is hard to describe in the footnote – but ask me if you have questions about this
[6]70 islands in old versions of PAUP. 58 islands on the version of PAUP posted June/2011
[7]1.08 seconds on my laptop
[8]147,531 rearrangements in old versions of PAUP. 155,616 rearrangments on the version of PAUP posted June/2011
[9]4 islands in old versions of PAUP. 3 islands on the version of PAUP posted June/2011
[10]two at 5689, one at 5693 and one at 5697 (the version of PAUP posted June/2011 does not find the tree with score 5693 for this seed)
[11]8 seconds on my laptop
[12]5,023,936 rearrangements in old versions of PAUP. 3,960,984 rearrangements on the version of PAUP posted June/2011
[13]4 islands in old versions of PAUP. 3 islands on the version of PAUP posted June/2011

- *What are the scores of the trees in each island?*[14]
- *How long did the search take?*[15]
- *How many rearrangements were tried?*[16]
- *How many trees are currently in memory (use the* `treeinfo` *command)?*[17]
- *Has PAUP* saved trees from all islands discovered during this search? (Hint: compare "Number of trees retained" to the sum of the "Size" column in the Tree-island profile.) Do you know why PAUP* saved the number of trees that it did?*[18]

  Wondering about that warning "Multiple hits on islands of unsaved trees may in fact represent different islands"? When PAUP* encounters a new island, it will find all trees composing that particular island in the process of branch swapping. Thus, if (in a new search) it encounters any trees already stored in memory, it knows that it has hit an island that it found previously. Note that it would be pointless to continue on this tack, because it will only find all the trees on that island again. For trees retained in memory, PAUP* can keep track of which island they belong to (remember that it is possible for trees with the same parsimony score to be in different tree islands!). But for trees that are not retained in memory, PAUP* only knows that it has encountered an island of trees having score X; it has no way of finding out how many islands are actually represented amongst the trees having score X.

If you want any of these commands to happen whenever you execute this file, then you can simply add the commands that you typed into the PAUP block of `run.nex`, add a semicolon after the command, and save the file.

[14] two at 5689, two at 5693 and one at 5697 (the version of PAUP posted June/2011 does not find the tree with score 5693 for this seed)

[15] 9 seconds on my laptop

[16] 14,790,674 rearrangements in old versions of PAUP. 10,698,858 rearrangements on the version of PAUP posted June/2011

[17] two

[18] No, it only save the trees from the best islands

Some of the distance methods in PAUP* and FastME

The goal of this part of the lab exercise is to show you how to conduct distance-based analyses in PAUP* and FastME.

Basic distance analyses in PAUP*

1. We will use the same `angio35b.nex` file that we used for the parsimony part of the lab.

2. Use a text editor to create a new file. Save it as `rund.nex` in the same directory as the `angiob.nex` file. This new file will be a NEXUS file that contains the PAUP block with the commands for PAUP. Here are the commands:

   ```
   #NEXUS

   begin paup;
     execute angio35b.nex;
     dset dist=abs;
     delete 5-.;
     exclude missambig;
     showdist;
   end;
   ```

   This file will tell PAUP* to:

   - use the absolute number of nucleotide differences between taxa as the distance measure (`dset dist=abs`);
   - delete all taxa except the first four (`delete 5-.`);
   - exclude all sites that have missing or ambiguous data (`exclude missambig`); and
   - show the distance matrix (`showdist`)

   Save the `rund.nex` file. Then execute it PAUP*, and examine the output.

3. **Calculate p-distances and JC distances** To see the distances as a proportion of sites that differ for the sequences just change the distance measure from `abs` to `p` and re-execute `rund.nex`. Examine the resulting data matrix, change the distance correction in `rund.nex` from `p` to `jc` to tell PAUP to use the Jukes-Cantor distance (this is the simplest model for correcting distances – Jeff will discuss the models tomorrow, but for this exercise it is only necessary to know that this is the name of a model used to correct the observed distances for multiple hits). Re-execute the file.

   - *How do the JC distances compare with the p-distances? Does the ordering of distances change?*[19]
   - *You have seen two distance measures that PAUP* can calculate, but how could you get a list of all the distance measures it can compute?*[20]

[19] The ordering does not change, but the JC distances are larger (and the larger *p*-distances are corrected more when you use the model-based correction).

[20] `dset ?`

4. Execute the `dset ?` command to see all of the distance settings that are available in PAUP. If you are confused by an option, you can check it out by downloading the PAUP manual from: http://paup.csit.fsu.edu/Cmd_ref_v2.pdf (or by asking me about an option).

5. **Estimate edge lengths using weighted least squares** Next, perform a search using weighted least squares (weighted usually implies that the power is 2 in the denominator of the sum-of-squares formula). Add the following line to your `run.nex` file, just below the `execute angiob35.nex;` line:

   ```
   set criterion=distance ;
   ```

   This command tells PAUP* to use the distance optimality criterion specified by the `objective` option of the `dset` command during the search. If you were to leave this out, PAUP* would use the default optimality criterion (parsimony). Now issue the command `dset ?` in PAUP* to get a listing of the current values of all distance settings.

   - What is the current setting for the `power` option? If your answer is not 2, then add this line to your paup block, below the `execute` command: `dset power=2;`
   - What is the current setting for the `objective` option? If your answer is not `lsfit`, then add this line to your paup block, also below the `execute` command: `dset objective=lsfit;`

   Finally, add the following two lines to the end of your paup block to perform the search and show the resulting best tree, including the estimated branch lengths:

   ```
   alltrees;
   describe 1 / brlens;
   ```

6. Save and re-execute the file.

7. Look for the line that begins “Weighted sum-of-squares =”. This is the least-squares score for the tree. In its output, PAUP* also gives you the score that would have been used were we using the minimum evolution criterion. *Can you find it?* Ask for help if you need to; PAUP* does not make this obvious. Write the value down for comparison with the value you will obtain in the next section.

8. **Searching under the minimum evolution criterion**. Before moving on to the next exercise, repeat the above search using the minimum evolution (or ME) criterion. To do this, you will need to add the `objective=ME` option to your `dset` command (be sure to remove the previous `dset objective=lsfit` if you had to put it in) and re-execute `rund.nex`.
   Is the result what you expected based on your answer to the last question in the previous section?[21]

Felsenstein zone example from lecture (optional)

You can download the file that I showed the “birds-eye view” of during lecture from:
http://www.people.ku.edu/ mtholder/848/data/fzone_sim.nex
It is an example of a “Felsenstein-zone” tree in which the taxa with long branches are not sister to each other.

1. Use PAUP* to find the least-squares tree for this data set using the p-distances.

[21]Hopefully. (Note: it is ok if the results differ slightly in the 5th. decimal place.)

2. *Is the tree the true tree, or the "long-branch attraction" tree preferred?*[22]
3. The data was simulated using the Jukes-Cantor model. Tell PAUP* to use the JC distance correction and do another search.
4. *Is the tree the true tree, or the "long-branch attraction" tree preferred?*[23]

Compare NJ with a comparable heuristic search (optional)

In this exercise, you will conduct a Neighbor-joining (NJ) analysis using JC distances and compare this with an heuristic search using the minimum evolution criterion. The goal of this section is to demonstrate that it is possible for heuristic searching to find a better tree than NJ, even using (almost) the same optimality criterion.

1. Please quit PAUP* and start it again. The reason for this will be made clear later, but mainly the purpose is to return all settings to their default values.
2. Put the commands below in a paup block in a new file. Note that we are again using the `angio35b.nex` file:

   ```
   execute angio35b.nex;
   dset distance=jc objective=me;
   nj;
   dscores 1;
   set Criterion = dist;
   hsearch start=1;
   dscores 1;
   ```

3. *What is the minimum evolution score for the NJ tree?* (scroll down from the beginning of the PAUP* output looking for the phrase "ME-score" right after point where the NJ tree is displayed)
4. *What is the minimum evolution score for the tree found by heuristic search starting with the NJ tree?*
5. *What is wrong? Why is the minimum evolution score of the heuristic search worse than that of the starting tree?* (Hint: take a look at the "Heuristic search settings" section of the output.)
6. Once you have figured out what is going on (ask me for help if you are stumped), fix your paup block and re-execute the file.
7. In your reanalysis, you should find that the heuristic search starting with the NJ tree found a better tree according to minimum evolution than NJ. Neighbor-joining is very popular, but you should be wary of NJ trees involving large numbers of taxa. This analysis involved 35 taxa; for problem of this size or larger, it is almost certain that NJ will not find the best tree.
8. *How much do the trees differ from each other?* To figure out, we'll need to get PAUP* to save the nj tree so that when we do the search we do not lose the NJ tree. Change your command file to say:

[22]the "long-branch attraction" tree
[23]the true tree

```
execute angio35b.nex;
set Criterion = dist;
dset distance=jc objective=me;
nj;
dscores 1;
savetree file = nj.tre ;
hsearch start=1;
dscores 1;
gettrees file = nj.tre mode = 7;
treedist ;
```

Here is the explanation:

- The `savetree` command writes the tree to a file (in the newick representation).
- The `gettrees` command reads the nj tree back into memor.
- The `mode=7` option to the `gettrees` command means "read the trees from the file, but do not throw away the trees that are currently in memory
- After `gettrees mode=7` command, you'll have the ME tree and the NJ tree in memory.
- The `TreeDist` command calculates tree-to-tree distances. The default is the symmetric-difference – the number of edges in tree #1 that are not in tree #2 plus the number of edges in tree #2 that are not in tree #1. It would be 0 if the trees were identical.

Balanced minimum evolution search in FastME

FastME is a program written by Desper and Gascuel. You can download it from
http://www.atgc-montpellier.fr/fastme/
Among other analyses (such as BIONJ, an algorithm that is similar to NJ, but does a better job with highly divergent sequences), FastME implements fast NNI searching under the balanced minimum evolution criterion. NJ is a quick and dirty search under this criterion, and FastME can do branch swapping to find even better trees. The program produces trees very quickly, and this is best demonstrated on large datasets.

1. Download 567Tx2153C.nex from
   http://www.people.ku.edu/∼mtholder/848/data/567Tx2153C.nex
   to get a large (567 taxon, 2153 character) data file.
2. FastME does not analyze sequences directly. Instead we have to give it a distance matrix. We will use PAUP* to create the input distance matrix.
3. Execute the 567Tx2153C.nex file in PAUP*.
4. Tell PAUP* to use the JC distance
5. Export the data file in a format that FastME can read using the command:
   `savedist format = phylip triangle = both diagonal file=dists.txt`
6. While you have PAUP open, do a NJ search, score the tree using minimum evolution, ordinary (unweighted) least squares, and weighted least squares.

7. Conduct a search under the weighted least squares criterion using the command: `HSearch NoMulTrees;` Score this tree under minimum evolution, ordinary (unweighted) least squares, and weighted least squares and note the amount of time the search took.
8. Now we will run FastME on the distance matrix. Open a command terminal and change the working directory of the terminal to the directory where you saved the `dists.txt` file. To invoke FastME, you simply type its name. On Mac, the command is `FastME_MacOS` (I must admit that I've never run it on Windows).
9. The program should prompt you for the name of the input file. At this point enter `dists.txt` and hit return. You should see a menu of options that let you choose what criterion to optimize and which algorithm to use. Conduct a Balanced Generalized Minimum Evolution search using the balanced NNI search. The program exits when it is finished (and it won't take long).
10. It should produce two output files: `dists.txt_stat.txt` and `dists.txt_tree.txt`. Open both in a text editor.
11. To get the tree into PAUP* we will have to change the tree file into a NEXUS file. Fortunately all you have to do to accomplish this is add:

    ```
    #NEXUS
    begin trees;
    tree fastme = [&U]
    ```

    to the file before the parenthetical notation, and

    ```
    end;
    ```

    to the end of the file.
12. Execute the `567Tx2153C.nex` file and set the distance back to the JC corrected distance.
13. Use the `GetTrees` command to do get the FastME trees into memory in PAUP*. Score this tree under minimum evolution, ordinary (unweighted) least squares, and weighted least squares and note the amount of time the search took.
14. How did the tree from FastME compare to the trees that you obtained by searching in PAUP? (recall that balanced minimum evolution is not the same as minimum evolution or least-squares, so it is not too surprising if you get different trees).

PAUP* Lab

Note: This computer lab exercise was written by Paul O. Lewis. Paul has graciously allowed Mark Holder to use and modify the lab for the Summer Institute in Statistical Genetics. Thanks, Paul!

In this computer lab you will learn the basics of using the computer program PAUP* for phylogenetic analyses of nucleotide sequences. Versions of PAUP* exist for several different operating systems (MacIntosh, Windows, Linux, etc.), with the MacIntosh version being the most flexible and user-friendly. We will be using the Windows version today. The graphical user interface (GUI) of this Windows version is not as well developed as the GUI for the MacIntosh version, but it is exactly the same program and produces results that are identical to the MacIntosh version.

The PAUP* Home Page is the best place to go for continuing updates on the progress being made toward the final release, and for information about purchasing the program: `http://people.sc.fsu.edu/~dswofford/paup_test/`

You can work through this tutorial at your own pace, asking questions whenever something needs to be clarified. Please let us know if you think another approach would be better, and if anything about this tutorial is unclear. The goals for this tutorial are to:

- Become familiar with the NEXUS data file format used by PAUP* (as well as several other prominent phylogeny programs such as Mesquite and MrBayes)
- Learn how to conduct various types of searches (exhaustive, branch-and-bound, heuristic using NNI and TBR branch swapping, and algorithmic approaches such as star decomposition and stepwise addition)
- Learn how to set up PAUP* to perform analyses under several different optimality criteria (maximum parsimony, minimum evolution, least squares, and maximum likelihood)
- Learn how to set up PAUP* for several different nucleotide substitution models, and to obtain maximum likelihood estimates of parameters of these models
- Learn how to create PAUP blocks in the data file so that analyses can be performed in batch mode (also learn why you might want to do this)

PAUP* Tutorial

Questions that you should be able to answer from looking at the output are in *italics*. Answers to the questions are provided in footnotes. If you do not understand one of these questions, or need help figuring out the answer, please do not hesitate to raise your hand.

About the data file

The tutorial uses one data file, `algae.nex`, which has been provided to you. This data set is distributed as one of the sample files for the program SplitsTree (`http://www.splitstree.org/`). It contains 16S rRNA sequences for a cyanobacterium (*Anacystis*), a chromophyte alga (*Olithodiscus*), a euglenoid protist (*Euglena*), and six green plants, including two green algae (*Chlorella* and *Chlamydomonas*), a liverwort

(*Marchantia*), a monocotyledonous angiosperm (*Oryza*, rice) and a dicotyledonous angiosperm (*Nicotiana*, tobacco).

This data set was used in a 1994 paper by Lockhart et al. to show how common models used in reconstructing phylogenies fail when confronted by convergence in nucleotide composition. The problem is that the common models assume stationarity of the substitution process, which leads to the assumption that base frequencies do not change across the tree. Thus, things can go wrong when the base frequencies do change from lineage to lineage, and things can go really wrong when unrelated groups tend to have similar base compositions. In this case, *Euglena* should group with the green plants because its chloroplast (whence the 16S rDNA is obtained) is homologous to green plant chloroplasts. However, as you will see, it has a strong tendency to group with the unrelated chromophyte *Olithodiscus* because of similarities in base composition. The complete reference to the Lockhart paper is

Lockhart, P. J., M. A. Steel, M. D. Hendy, and D. Penny. 1994. Recovering evolutionary trees under a more realistic model of sequence evolution. *Molecular Biology and Evolution* **11**: 605-612.

Tutorial begins here

1. Start PAUP* by double-clicking its icon. After PAUP* starts, it will present you with an **Open** dialog box. Navigate to the file `algae.nex` and click the **Open/Execute** button when the file's name has been selected.

2. Before doing any analyses, let's take a look at the data file PAUP* just executed [1] Open the `algae.nex` file for editing by choosing **File** | **Open...** from the main menu (**Ctrl-O** does the same thing), clicking the **Edit** radio button in the **File Open Mode** group, selecting the file name, and finally clicking the **Open/Edit** button.

   The Nexus data file format has been adopted by several phylogenetic analyses programs, including PAUP*, MacClade, Mesquite, SplitsTree, TreeView, and MrBayes, among others. Nexus data files always begin with `#nexus`, and the remainder of the file is divided into units known as **blocks**. Nexus files are (for the most part) case-insensitive, so `#nexus`, `#Nexus` and `#NEXUS` are synonyms. This file has two blocks: a **taxa block** and a **characters block**. Each block begins with the keyword **begin** and ends with the keyword **end**. Each block comprises **commands**, all of which end in a semicolon (;). Note that each block automatically has two commands: the **begin** command and the **end** command. Some commands are quite long, taking up many lines in the file (e.g., the **matrix** command in the characters block), but the extent of each command can be surmised by simply looking for that terminating semicolon. A mistake made by most everyone when first constructing a Nexus data file is to forget to end every command with a semicolon. If you do this, PAUP* will report an error when attempting to read in the data file.

   What are the four commands comprising the TAXA block? [2]

   Nexus files can contain comments. Comments are text surrounded by square brackets. Comments that you wish to have printed out in the output look like this:

[1] PAUP* uses the term **execute** to mean reading a data file for the purpose of storing the data contained therein. The term **edit** is used for the opening of a data file when the purpose is to view/modify its contents and not to perform analyses.

[2] The four commands comprising the TAXA block are: (1) "begin taxa;"; (2) "dimensions ntax=8;"; (3) "taxlabels [1] Tobacco [2] Rice ... [8] Olithodiscus;" and (4) "end;".

```
[!This is a printed comment]
```

If that initial exclamation point (!) is missing, PAUP* will simply ignore the comment entirely. *Can you find the single printed comment both in the data file and in PAUP*'s output?* [3]

Here is a brief explanation of some of the commands present in this data file:

Command	Meaning
`dimensions ntax=8;`	Data are provided for eight taxa
`taxlabels Tobacco Rice ··· Olithodiscus;`	Provides names for the eight taxa
`dimensions nchar=920;`	There are 920 nucleotide sites for each sequence (taxon)
`format datatype=RNA missing=?  gap=- labels interleave;`	These are RNA sequences; the symbol `?` is used for missing data and `-` for gaps introduced for purposes of aligning the sequences; taxon labels are provided before each sequence; and the data are interleaved, which means that sequences are broken up into short segements for readability

For a complete reference to the Nexus data file format, please see the following paper:

Maddison, David R., Swofford, David L. and Maddison, Wayne P. 1997. NEXUS: an extensible file format for systematic information. *Systematic Biology* **46**: 590-621

Close the editor now by clicking on the button labeled with an × in the upper right-hand corner of the editor window.

3. **Parsimony Analysis.** For n taxa, there are

$$\frac{(2n-5)!}{(n-3)!\,2^{n-3}}$$

possible unrooted, fully-bifurcating, distinct tree topologies. For our 8 taxa, this formula produces $10,395$. This is not astronomical, so we will use an exhaustive search in combination with the maximum parsimony criterion for our first analysis. Type the following commands into the edit control near the bottom of PAUP*'s main window (you can either type each one in, pressing the enter key after each, or type them both in and press the enter key only after both have been entered – the semicolons serve to keep the commands separated):

```
set criterion=parsimony;
alltrees;
```

[3]The comment is "[! Example of RNA data]".

Note: the search status dialog box has a single button that says “Stop” while the search is in progress and “Close” when the search is done. If the button says “Close”, this means PAUP* is no longer doing any work, but is simply sitting there waiting for you to press the button!

The first command is not strictly necessary, since the parsimony criterion is selected by default in PAUP*; however, defaults can change, so it is wise to be explicit. Look at the output (you may have to scroll up some to see the beginning) and answer the following questions before proceeding:

How many tree topologies did PAUP look through? Was this what you expected? How many trees had the best score? How many steps longer than the most parsimonious tree(s) is the next best tree? How long, in your estimation, would it take to perform an exhaustive search for 16 taxa? (Note that the answer is not twice the amount of time required for 8.)* [4]

Use the command `showtrees all;` to tell PAUP* to show you all trees currently stored in memory. Although these are unrooted trees, they appear to be rooted at tobacco and rice, which is not a very useful rooting. Use the following command to tell PAUP* to root trees shown from now on using *Anacystis nidulans* as the outgroup:

```
outgroup Anacystis_nidulans;
```

Note the use of the underscore character (_) to make *Anacystis nidulans* a single word. PAUP* uses blank spaces (including tab and newline characters) to delimit individual items (called words or tokens). If we had left out the underscore character, PAUP* would have attempted to make both *Anacystis* and *nidulans* outgroups, but would have run into problems since *nidulans* is not a taxon. Another way to handle embedded blank spaces is to enclose the entire token in single quotes (e.g., `'Anacystis nidulans'` would also have worked).

One of the interesting features of this data set is that many analysis methods do not produce an estimated tree in which all 6 chlorophyll-b-containing taxa are together. There is considerable evidence from many sources that this should indeed be the case, so throughout this tutorial be on the watch for methods that produce a tree in which this **chlorophyll-a/b clade** is separated by an internal branch from the other two taxa: *Olithodiscus* (a chlorophyll-a/c-containing chromophyte alga) and *Anacystis* (a cyanobacterium containing only chlorophyll-a plus phycobilin accessory pigments).

Does standard (Fitch) parsimony produce the chlorophyll-a/b clade? If not, what taxon is separated from the rest of the chlorophyll-a/b clade? [5]

4. **Branch-and-bound.** Although we have already done an exhaustive search and thus have no need to perform a branch-and-bound search, feel free to use the command below to perform a branch-and-bound search anyway.

   ```
   bandb upbound=470;
   ```

   Note that an upper bound was supplied in this case. This upper bound is the score of some tree (not necessarily the best) containing all of the taxa. If you leave out the upper bound specification, PAUP* will compute an initial upper bound for you using stepwise addition. Notice how many fewer trees have to be evaluated when you used a branch-and-bound search.

[4]PAUP* looked through 10395 trees, which is the number of unrooted binary trees for 8 taxa. The best score was 411, and 2 trees had this score. The next best tree is 414 steps long, and only 1 tree had this score. On my laptop, it required 0.04 seconds to evaluate the 10,395 possible trees for 8 taxa, but there are literally trillions of possible trees for 16 taxa ($27!/(13!2^{13})$ = 213,458,046,676,875), so it would take some 26 years to score all possible trees for 16 taxa.

[5]*Euglena* is grouped with *Olithodiscus*, not with the other a/b taxa

5. **Distance Analyses: Neighbor-joining.** To obtain the neighbor-joining (NJ) tree, simply issue the command `nj`. This should produce the same tree as parsimony, although it will look a bit different since PAUP* shows the tree as a **phylogram** (with branch lengths proportional to the estimated number of substitutions) rather than the **dendrogram** format (the default for the `showtrees` command used earlier). To check that it is indeed equivalent to one of the most-parsimonious trees, issue the `pscores all` command to obtain parsimony scores for all trees in memory (but there should be just 1 tree in memory). The treelength shown should match the one obtained earlier in our parsimony searches.

6. **Distance Analyses: NNI heuristic search using ME criterion and JC distances.**

   To save time and keep this lab from being too tedious, I am combining several goals in each analysis. Note that PAUP* provides considerable flexibility in allowing you to mix-and-match optimality criteria, search strategies, and substitution models. The combinations used in this tutorial should not be viewed as necessarily the "best" combination.

   In this case we will combine a heuristic search strategy (using NNI branch swapping) with the minimum evolution (ME) optimality criterion and will use the Jukes-Cantor model for obtaining the pairwise distance matrix. Here are the commands necessary:

```
set criterion=distance;
dset distance=jc objective=me;
hsearch swap=nni;
```

   The `dset` command is used for changing the **distance settings**. You may be wondering at this point "How do I know which distance settings to change?" To get PAUP* to give you a summary of any command, simply type the command's name followed by a space and then a question mark:

```
dset ?;
```

   Doing so provides a list of all the options available for that command as well as their current settings.

   What is the only other option (besides `ME`*) for the setting called* `objective`*? How would you instruct PAUP* to use HKY85 distances?* [6]

   Get PAUP* to show you a phylogram of the best tree found using the command

```
describetree 1 / plot=phylogram;
```

   Does this distance tree separate out the chlorophyll-a/b clade? What method did PAUP use for obtaining the starting tree? Use the method described above (i.e., the question mark approach) to obtain a list of options for the* `hsearch` *command. How would you instruct PAUP* to use the stepwise addition method for obtaining the starting tree?* [7]

   Note that NJ is an approximation to the algorithmic approach known as **star decomposition** in conjunction with the minimum evolution (ME) optimality criterion. Thus, an analysis similar to neighbor-joining but which is not approximate could be conducted as follows:

[6]The only other option for `objective` is `LSFit`. To use HKY85 distances, you would issue the command `dset distance=HKY85`

[7]No, *Euglena* is still with *Olithodiscus*. To use stepwise addition to obtain the starting tree, you would issue the command `hsearch start=stepwise`

```
set criterion=distance;
dset objective=me;
stardecomp;
```

Note that the NJ command uses the current distance settings to determine which model to use in computing the pairwise distance matrix.

7. **Distance Analyses: TBR heuristic search using LS criterion and LogDet distances.**

   This distance analysis will use a different optimality criterion (least-squares, or LS, rather than ME) and a different model for obtaining the pairwise distances (LogDet rather than JC). We will also use TBR branch swapping rather than NNI.

   ```
   dset distance=logdet objective=lsfit;
   hsearch swap=tbr;
   describetrees 1 / plot=phylogram;
   ```

   Note that this time the chlorophyll-a/b clade is intact! What is different about this analysis? The most important difference is the model used to compute the pairwise distances. LogDet is a very good model to use when nucleotide composition varies across the tree. Use PAUP*'s `basefreq` command to examine the nucleotide composition of all the sequences in this data file. Since *Euglena* has been the taxon jumping out, and it tends to join with *Olithodiscus* when it is misbehaving, look specifically at the nucleotide composition of *Euglena* and *Olithodiscus* compared to everything else. *Are these two taxa similar to each other in nucleotide composition?* If so, then most methods may be placing them together simply because of the similarity in their nucleotide composition.

8. **Likelihood Analyses: HKY85 model combined with stepwise addition.** For this analysis, we will use the maximum likelihood criterion and obtain a tree using the stepwise addition method. Stepwise addition is normally not considered an end in itself; it is most often used to obtain a **starting tree** for input into heuristic searches. Likelihood runs can be very time consuming if the number of taxa is large, and it is nice to be able to separate the process of obtaining the starting tree from the heuristic search proper. Stepwise addition trees can also come in handy for estimating parameters of the substitution models. These parameters can then be fixed at their estimated values for the duration of the search rather than estimated anew for each tree examined during the search. The latter is of course better, but the former may be the only practical course of action.

   ```
   set criterion=likelihood;
   lset nst=2 basefreq=empirical variant=hky tratio=estimate rates=equal;
   hsearch start=stepwise addseq=random swap=none nreps=1;
   ```

   This set of commands requires some explanation. PAUP* does not allow you to specify the substitution model by name as it does for distance analyses. You must specify instead the *characteristics* of the model you wish to use. Let's look first at the line specifying the model, then we will tackle the line specifying the search strategy:

   ```
   lset nst=2 basefreq=empirical variant=hky tratio=estimate rates=equal;
   ```

This line provides **likelihood settings** corresponding to the HKY85 model. This model allows **unequal base frequencies**, and `basefreq=empirical` instructs PAUP* that the **empirical base frequencies** should be used. The empirical base frequencies are simply the base frequencies computed from the data matrix without reference to any tree topology. These base frequencies work about as well as the maximum likelihood estimates of the base frequencies, which PAUP* can also compute (using `basefreq=estimate`), but estimating the base frequencies will add a considerable amount of time to any analysis. In either case, the base frequencies employed will almost certainly be unequal, which fits the HKY85 model and excludes models such as JC and K2P which assume all bases are equally frequent (i.e., $\pi_A = \pi_C = \pi_G = \pi_T = 0.25$). The `nst=2` part tells PAUP* to use a model employing 2 substitution rate parameters (the K2P, F84, and HKY85 models all allow transitions to occur at a potentially different rate than transversions, hence two substitution rates). The `variant=hky` statement is necessary since both the F84 model and the HKY85 model allow unequal base frequencies and transition/transversion bias. The `tratio=estimate` instructs PAUP* to estimate the transition/transversion rate for every tree. Finally, the `rates=equal` statement says that PAUP* is to assume that all sites evolve at the same overall substitution rate.

```
hsearch start=stepwise addseq=random swap=none nreps=1;
```

Specifying `swap=none` prevents PAUP* from actually carrying out the heuristic search. Instead, it will create a stepwise addition starting tree (`start=stepwise`) using a random sequence to determine the order in which taxa are added (`addseq=random`) and then simply stop, holding that stepwise addition tree in memory. The `nreps=1` statement is necessary to prevent PAUP* from performing its default number of 10 independent search replicates.

Run the `lset` and `hsearch` commands. Then use `showtrees` to see the estimated phylogeny and use the following command to obtain the maximum likelihood estimate of the transition/transversion ratio and rate ratio:

```
lscores 1;
```

This command instructs PAUP* to compute the likelihood score [8] for the 1st. tree in memory. This forces PAUP* to also spit out estimates of all parameters you asked it to estimate, which in this case is just the transition/transversion ratio.

What model did PAUP say it used for this analysis? What is the maximum likelihood estimate of the transition/transversion ratio and rate ratio? Did this analysis result in an estimated phylogeny that separates out the chlorophyll-a/b clade?* [9]

You probably noticed that this analysis took considerably longer than any analysis performed thus far, even though we specifically avoided doing a search! The main reason this one took so long was because we instructed PAUP* to estimate the transition/transversion ratio (tratio) for every tree examined during the analysis. Let's fix the value of tratio to the maximum likelihood estimate just obtained and try again to see how much faster things go:

```
lset tratio=previous;
hsearch;
```

[8] There are corresponding commands for obtaining scores for the parsimony (`pscores`) and distance (`dscores`) criteria.

[9] PAUP* reports "These settings correspond to the HKY85 model." The transition/transversion ratio was estimated to be 1.910491, whereas the transition/transversion rate ratio was estimated to be 3.637613. No, this analysis did not separate the chlorophyll-b-containing taxa from the other two.

In my case, it took about 4.5 times longer to do the search when estimating the tratio for every tree examined, so fixing the tratio at some reasonable value sped up the analysis considerably. It is common to estimate parameters on a tree that can be obtained quickly (e.g., a NJ tree), then fix the values of those parameters for the duration of a search. If you employ this strategy, it is probably a good idea to re-estimate the parameters after the search to make sure they are not appreciably different. To be safe, it is best to repeat the search using the newer estimates. If the second search returns the same tree as the first search, then you know that the initial estimates were reasonable.

9. **Maximum Likelihood Analyses: Using the GTR model.**

The HKY85 model is rather inflexible in that it allows for only two classes of substitutions – transitions and transversions. If the two types of transitions (purine $\leftrightarrow$ purine vs. pyrimidine $\leftrightarrow$ pyrimidine) occur at different rates, or if the four types of transversions occur do not all occur at the same rate, the HKY85 model might not capture important aspects of the evolution of the sequences under study. The General Time Reversible, or GTR, model allows the six possible classes of substitutions (i.e., $A \leftrightarrow C$, $A \leftrightarrow G$, $A \leftrightarrow T$, $C \leftrightarrow G$, $C \leftrightarrow T$, and , $G \leftrightarrow T$) to all occur at potentially different rates. This adds five extra parameters to the JC model rather than just the one added by the HKY85 model, and estimating all of these for each tree examined would be quite time consuming (although not that impractical for this data set because of the relatively small number of taxa). Our approach will be to estimate the five extra relative rate parameters using the tree currently in memory, then fix these values for purposes of conducting the heuristic search.

To set up the GTR model and estimate the relative rate parameters, use these commands:

```
lset nst=6 rmatrix=estimate;
lscores 1;
```

The output should show something like this:

```
Tree             1
------------------
-ln L   3251.02830
Rate matrix R:
  AC       0.62353
  AG       1.87194
  AT       0.82691
  CG       0.32529
  CT       3.68057
  GT       1.00000
```

The **R-matrix** contains the estimates of the relative rates (rows and columns both are in the order A, C, G, T). PAUP* always reports the rate of the $G \leftrightarrow T$ change as 1. Since these are relative rates having meaning only in comparison to each other, any one of them may be set to some arbitrary value or some overall constraint may be applied (i.e., their mean could be made 1.0). In PAUP*, the former method is used and the last one is arbitrarily set to the value 1. The way to interpret these relative rates is best demonstrated by example: all other things being equal, the rate at which $C \leftrightarrow T$ changes occur is 3.68 times the rate at which $G \leftrightarrow T$ changes occur, and $A \leftrightarrow C$ changes are occurring at about a third the rate of $A \leftrightarrow G$ changes (0.62353/1.87194 = 0.33309).

The **Q-matrix** takes account of the effects of the base frequencies. To see the Q-matrix, re-issue the `lscores` command, but this time with a couple of options:

```
lscores 1 / longfmt showqmatrix;
```

The output should now show the R-matrix (displayed in matrix format rather than a simple list) followed by the Q-matrix. In the GTR model, the rates at which the various changes occur are a function not only of these relative rates, but also involve the frequency of the base being substituted *to*. For example, $Q_{TC} = \mu R_{TC} \pi_C$, where μ is the overall rate of substitution, R_{TC} is the relative rate of the $T \leftrightarrow C$ substitution class (from the R-matrix), and π_C is the frequency of C. Thus, the rate at which $T \leftrightarrow C$ changes occur is influenced by the frequency of the base C: if Cs are common, this change will occur at a higher rate than if Cs are rare.

Now fix the rmatrix values and conduct an heuristic search using the following commands:

```
lset rmatrix=previous;
hsearch start=1 swap=tbr;
describe 1 / plot=phylogram;
```

Note that we do not need to obtain a starting tree, since we have already done that, so we tell PAUP* to simply begin with the first (and only) tree currently in memory (`start=1`) and conduct a heuristic search using TBR branch swapping.

Does the tree that PAUP estimates using the GTR model split the chlorophyll-a/b clade from the other two taxa?* [10]

10. **Maximum Likelihood Analyses: Adding Discrete Gamma Rate Heterogeneity.**

We have one more important feature to add to our model. Among-site rate variation is pronounced in most sequence data sets. A common way to model such heterogeneity in rates across sites is through the use of the discrete gamma model. This model assumes that the relative rates [11] are gamma-distributed with a shape parameter usually referred to as α. If the shape parameter is large (i.e., close to ∞), then the relative rates will all be clustered around their mean, 1.0, meaning that there is essentially **rate homogeneity**. If α is very low (i.e., close to 0.0), the relative rates are extremely dispersed with most near zero but a few very high values. This represents the case of high **rate heterogeneity**. Thus, using a gamma distribution to model rates allows us to capture much of the possible range of relative rate distributions with the addition of only one more parameter (α) to our model.

Our first step will be to estimate the amount of rate heterogeneity (and re-estimate the rmatrix, since it will be different under a rate heterogeneity model) using the tree already in memory:

```
lset rates=gamma ncat=4 shape=estimate rmatrix=estimate;
lscores 1;
```

To see graphically what the gamma distribution with this shape looks like, issue the gammaplot command:

```
gammaplot;
```

[10] No, *Euglena* is still stuck on *Olithodiscus*

[11] Note that these relative rates are different from those just discussed. Here we are referring to the relative rates at which different sites evolve; before, we were referring to the relative rates of different classes of substitutions

The shape of the distribution makes it obvious that this is rather extreme rate heterogeneity (low rate heterogeneity would be indicated by a sharp peak over the value 1.0). Now fix both the rmatrix and gamma shape parameters at their estimated values and conduct a new search, starting from a random stepwise addition tree and using TBR branch swapping.

```
lset shape=previous rmatrix=previous;
hsearch start=stepwise addseq=random swap=tbr;
describe 1;
```

This time you should see that chlorophyll-a/b clade return, so rate heterogeneity is obviously an important factor in the evolution of these sequences. The question "How important a factor is rate heterogeneity?" now arises. Is it more important, for example, than allowing for six different substitution rates? It turns out that for all but the simplest models (i.e., Jukes-Cantor and F81), allowing rate heterogeneity will produce the chlorophyll-a/b clade! To see this, here are the commands for setting up the Kimura 2-parameter model (K2P) with rate heterogeneity. This first obtains a stepwise addition tree using the plain K2P model, then estimates tratio and the gamma shape parameter on the stepwise addition tree, fixing the value of these two parameters before conducting a more thorough TBR search:

```
lset nst=2 basefreq=equal tratio=estimate rates=equal;
hsearch start=stepwise addseq=random swap=none nreps=1;
lset rates=gamma shape=estimate;
lscores 1;
lset shape=previous tratio=previous;
hsearch start=1 swap=tbr;
describe 1;
```

11. **Saving, viewing and printing trees using TreeView**

    TreeView is a free program written by Rod Page that can be downloaded from this web page:

    http://taxonomy.zoology.gla.ac.uk/rod/treeview.html

    TreeView provides a graphical interface for viewing, editing, and printing trees produced by PAUP* or other programs. This somewhat makes up for the fact that PAUP*'s tree printing machinery has not yet been completed for the Windows version.

    Alternatively, you can use FigTree by Andrew Rambaut:
    http://tree.bio.ed.ac.uk/software/figtree/

    To illustrate the use of TreeView (or FigTree) in conjunction with PAUP*, we will save a tree within PAUP* and then open and view it in TreeView. In PAUP*, issue this command to save the tree currently stored:

    ```
    savetrees file=test.tre brlens;
    ```

    This will save the tree currently stored in memory to the file `test.tre`. The `brlens` keyword tells PAUP* to store the branch lengths as well as the tree's topology. Start TreeView, and select `test.tre` in the dialog box that appears upon choosing the **File** | **Open...** menu item. Once the treefile has been opened, try switching to phylogram view (menu choice **Tree** | **Phylogram**) to take advantage of the fact that the branch lengths were stored. You can either print the tree if a printer is connected, or save the tree as a Windows metafile (which can later be imported into a Word document, for example). To save the tree as a Windows metafile, choose **File** | **Save as graphic...**.

12. **Constructing and Using PAUP Blocks.**

All commands that can be entered from the keyboard can also be placed in a NEXUS file in the form of a PAUP block. For example, the following PAUP block would enable us to automatically conduct the K2P analysis (see end of last step in the tutorial):

```
begin paup;
  log file=algae.log start replace;
  set autoclose=yes;
  execute algae.nex;
  outgroup Anacystis_nidulans;
  set criterion=likelihood;
  lset nst=2 basefreq=equal tratio=estimate rates=equal;
  hsearch start=stepwise addseq=random swap=none nreps=1;
  lset rates=gamma shape=estimate;
  lscores 1;
  lset shape=previous tratio=previous;
  hsearch start=1 swap=tbr;
  describe 1 / plot=phylogram;
  log stop;
end;
```

The first line after the `begin paup` command starts a log file named `algae.log` to which all the output will be echoed. The `replace` keyword in the `log` command tells PAUP* to automatically (i.e., without asking) overwrite any existing file named `algae.log`. The command `set autoclose=yes` tells PAUP* that we would like the progress dialog box that appears whenever a search is initiated to close without our intervention after each of the searches are completed. The `execute` command executes the `algae.nex` data file, which is a necessary prerequisite to doing an analysis. The `execute` command is equivalent to inserting the entire text of the `algae.nex` file (except for the initial `#nexus` keyword) into this file in place of this line. The only other command you have not yet seen is the last one before the `end` command, which instructs PAUP* to close the log file.

Why would one wish to place all the commands for an analysis into a separate file? I find it very useful for the simple reason that it is very easy to forget exactly what analyses you performed for a particular paper, and it is even easier to forget which file names you used for each separate analysis. If you use PAUP blocks, then you can exactly repeat an analysis at a later time. Another use for PAUP blocks is to perform initial setup steps. For example, if you always use the likelihood criterion whenever you analyze a particular data file, and if you tend to always use the HKY85 model, you could have PAUP* switch to the likelihood criterion and the HKY85 model every time you execute the data file by inserting the following PAUP block at the bottom of the file:

```
begin paup;
  log file=logfile.txt start append;
  set criterion=likelihood;
  lset nst=2 basefreq=empirical tratio=estimate rates=equal;
end;
```

Note that this time we are placing the PAUP block into the actual data file. While this is legal, I try to stay away from starting analyses inside the data file - better to keep analyses in separate NEXUS files so that you do not accidentally start long analyses when your intention was only to open the data file.

In keeping with this suggestion, the above PAUP block does not perform any analyses, but it does start a log file and set up both the criterion and the model for you. This time I used the `append` keyword in the `log` command, which causes PAUP* to add new material to the end of the file `logfile.txt` if it already exists (`append` is safer than `replace` because existing files are never overwritten).

One further tip about PAUP blocks. I often create multiple PAUP blocks – one for each analysis done – in a single file. The ones that are currently not in use I simply "turn off" by changing the name slightly (for example, change the name from `paup` to `_paup`). If PAUP* does not recognize a block name, it skips the block and moves on to the next `begin` command. Thus, you can accumulate old PAUP blocks as a record of past analyses.

13. **Constructing Nexus Data Files and Finding Islands In Treespace**

Choose **File** | **New** from PAUP*'s menu (or press Ctrl-N) to open a new text editor window. Type the following small data set into the window, then choose **File** | **SaveAs...** from PAUP*'s menu (or press Ctrl-S) to save the file (call it `islands.nex`, or make up a name of your own).

```
#nexus

begin data;
  dimensions ntax=5 nchar=8;
  format datatype=dna;
  matrix
    A  A C G C A G G T
    B  A T G G T G A T
    C  G C T C A C G G
    D  A C T G T C G T
    E  G T T C T G A G
  ;
end;

begin paup;
  hsearch start=stepwise addseq=random nreps=100 swap=nni;
end;
```

After saving this file, use **File** | **Open...** (or press Ctrl-O) to execute the file. PAUP* should immediately perform the search since it was specified in a PAUP block in the data file itself. The `nreps=100` in the `hsearch` command tells PAUP* to perform 100 searches, each beginning with a (potentially [12] different random addition starting tree. Note that PAUP* was able to find both tree islands. This is the same data file presented in class as an example of tree islands. Look back at your notes for the figure showing the NNI connections between the trees, which shows that there are two islands in this treespace, each of which has two members.

Now redo the search but specify only one replicate. Type the following command into PAUP* command edit control at the bottom of the main window:

[12] I say potentially here since there are only 20 distinct random addition sequences for 5 taxa and thus there will be many replicates in which the random addition sequence is identical to that of previous replicates.

```
hs nreps=1;
```

Note that I have abbreviated `hsearch` as simply `hs` this time. PAUP* allows abbreviations of commands as long as enough letters are provided to distinguish the command from all other commands.

Now how many islands does PAUP find, and what are the sizes of these islands? Can you explain why PAUP* did not find all the islands this time?* [13]

Now redo the search once more, again specifying only one replicate, but this time perform an SPR search:

```
hs nreps=1 swap=spr;
```

Now how many islands does PAUP find, and what are the sizes of the islands? Can you explain why PAUP* obtained this result?*/footnoteOnly one island found, but it contains all four trees that were on two separate islands in the NNI search. Explanation: SPR search explores a different landscape in which there is no "valley" between these trees – it is able to easily hop over the divide that stopped NNI.

[13] It only finds a single island this time because, by definition, once it finds a tree in one island it will be unable to bridge the gap to find trees of equal score on other islands.

Tutorial Quick Reference

Here are the commands issued during the tutorial. This is useful if you want to proceed quickly through commands with which you are familiar, and then go back to the main tutorial when you hit something you don't understand.

Parsimony exhaustive search produces 2 trees of length 411 steps (p. 3)

```
set criterion=parsimony;
alltrees;
```

Set outgroup for display purposes (p. 4)

```
outgroup Anacystis_nidulans;
```

Branch-and-bound parsimony analysis yields same two trees of 411 steps (p. 4)

```
bandb upbound=470;
```

NNI search using JC distances produces tree with ME score 0.43085 (p. 5)

```
set criterion=distance;
dset distance=jc objective=me;
hsearch swap=nni;
```

Using question mark to get list of options for a command (p. 5)

```
dset ?;
```

Using describetree to show phylogram (p. 5)

```
describetree 1 / plot=phylogram;
```

Star-decomposition using the ME criterion yields tree with ME score 0.43085 (p. 5)

```
set criterion=distance;
dset objective=me;
stardecomp;
```

TBR search using LogDet distances and the least-squares criterion, yielding tree with score 0.03831 (p. 6)

```
dset distance=logdet objective=lsfit;
hsearch swap=tbr;
describetrees 1 / plot=phylogram;
```

Stepwise addition using likelihood and HKY model, yielding tree with score 3274.21265 (p. 6)

```
set criterion=likelihood;
lset nst=2 basefreq=empirical variant=hky tratio=estimate rates=equal;
hsearch start=stepwise addseq=random swap=none nreps=1;
```

Using lscores command to show MLEs of parameters, with ti/tv ratio 1.910491, kappa 3.637613, and the negative log-likelihood 3274.21265 (p. 7)

```
lscores 1;
```

Fixing parameter values using the previous option, search produces same tree with score 3274.21265 (p. 7)

```
lset tratio=previous;
hsearch;
```

Estimate parameters of the GTR model on the HKY tree in memory, yielding score 3251.02830 and rmatrix rAC=0.62353, rAG=1.87194, rAT=0.82691, rCG=0.32529, rCT=3.68057 and rGT=1.0 (p. 8)

```
lset nst=6 rmatrix=estimate;
lscores 1;
```

Use longfmt to show R-matrix and Q-matrix in matrix form (p. 8)

```
lscores 1 / longfmt showqmatrix;
```

TBR search using GTR model and starting with the HKY tree fails to find a better tree, score (3251.02830) identical to the HKY tree (p. 9)

```
lset rmatrix=previous;
hsearch start=1 swap=tbr;
describe 1 / plot=phylogram;
```

Estimate among-site rate heterogeneity for current tree, MLE of shape parameter is 0.255221 (p. 9)

```
lset rates=gamma ncat=4 shape=estimate rmatrix=estimate;
lscores 1;
```

The gammaplot command shows you what the gamma density function looks like for the current shape parameter (p. 9)

```
gammaplot;
```

TBR search starting with stepwise addition and allowing rate heterogeneity in the GTR model yields tree with score 3155.85106 (p. 10)

```
lset shape=previous rmatrix=previous;
hsearch start=stepwise addseq=random swap=tbr;
describe 1;
```

TBR search using K2P model with rate heterogeneity. First tree obtained (using plain K2P model) has score 3178.17621, tratio 2.435876, and kappa 4.871753. The second search with rate heterogeneity yields a tree with score 3178.07881 (p. 10)

```
lset nst=2 basefreq=equal tratio=estimate rates=equal;
hsearch start=stepwise addseq=random swap=none nreps=1;
lset rates=gamma shape=estimate;
lscores 1;
lset shape=previous tratio=previous;
hsearch start=1 swap=tbr;
describe 1;
```

Save trees to file using the savetrees command (p. 10)

```
savetrees file=test.tre brlens;
```

Using PAUP blocks to automate analyses (p. 11)

```
begin paup;
  log file=algae.log start replace;
  set autoclose=yes;
  execute algae.nex;
  outgroup Anacystis_nidulans;
  set criterion=likelihood;
  lset nst=2 basefreq=equal tratio=estimate rates=equal;
  hsearch start=stepwise addseq=random swap=none nreps=1;
  lset rates=gamma shape=estimate;
  lscores 1;
  lset shape=previous tratio=previous;
  hsearch start=1 swap=tbr;
  describe 1 / plot=phylogram;
  log stop;
end;
```

Data set for seeing tree islands resulting from NNI search. Two islands are found, both with score 13, and each of size 2 trees (p. 12)

```
#nexus

begin data;
  dimensions ntax=5 nchar=8;
  format datatype=dna;
  matrix
    A  A C G C A G G T
    B  A T G G T G A T
    C  G C T C A C G G
    D  A C T G T C G T
    E  G T T C T G A G
  ;
end;

begin paup;
  hsearch start=stepwise addseq=random nreps=100 swap=nni;
end;
```

NNI search with one replicate only finds one of the two known islands (p. 12)

```
hs nreps=1;
```

SPR search finds all four tree even though one replicate specified (p. 13)

```
hs nreps=1 swap=spr;
```

Summary

In summary, and for easy reference, here are the commands necessary to set up many of the models and search strategies discussed. There are many options for most of the commands; remember that you can use the question-mark approach to obtain a listing of all the options for any given command.

Search Strategies

Search strategy	PAUP* commands
Stepwise addition (only)	`hsearch start=stepwise addseq=random swap=no;`
Star decomposition	`stardecomp;`
Exhaustive enumeration	`alltrees;`
Branch-and-bound	`bandb;`
Stepwise addition + NNI heuristic search	`hsearch start=stepwise addseq=random swap=nni;`
Stepwise addition + SPR heuristic search	`hsearch start=stepwise addseq=random swap=spr;`
Stepwise addition + TBR heuristic search	`hsearch start=stepwise addseq=random swap=tbr;`

Distance Analyses

To get this:	Use these PAUP* commands
Neighbor-joining	`nj;`

Minimum Evolution criterion	`set criterion=distance;`
	`dset objective=me;`

Least-squares criterion (unweighted)	`set criterion=distance;`
	`dset objective=lsfit power=0;`

Fitch-Margoliash criterion (weighted least-squares)	`set criterion=distance;`
	`dset objective=lsfit power=2;`

JC69 model	`set criterion=distance;`
	`dset distance=jc;`

F81 model	`set criterion=distance;`
	`dset distance=f81;`

F84 model	`set criterion=distance;`
	`dset distance=f84;`

HKY85 model	`set criterion=distance;`
	`dset distance=hky85;`

GTR model	`set criterion=distance;`
	`dset distance=gtr;`

LogDet model	`set criterion=distance;`
	`dset distance=logdet;`

ML distances[a]	`set criterion=distance;`
	`dset distance=ml;`

[a] If ML distances are desired, use the `lset` command to set up the desired maximum likelihood model (see tables that follow), but make sure "`set criterion=distance;`" is in effect when the analysis is started.

Likelihood Models

In each of these models containing extra parameters (e.g., transition/transversion ratio), I have provided for estimating these extra parameters in the commands. Remember that this will add a lot of time to the analysis, and for each of these, there is a way to either set the parameter to the previous value or to any particular value. For example, to set `tratio` to a value previously estimated, you can use `tratio=previous` rather than `tratio=estimate`. To set `tratio` to some particular value, say 2.5, use `tratio=2.5`. The `rmatrix` setting works a little differently since there are 5 values associated with the `rmatrix` setting. To set `rmatrix` to values previously estimated, use `rmatrix=previous`; however, to set the `rmatrix` to a set of five specific values, say 0.62, 1.87, 0.82, 0.32, and 3.68, use this format: `rmatrix=(0.62 1.87 0.82 0.32 3.68 )`.

Models Assuming Rate Homogeneity

Model	PAUP* commands

JC69	`set criterion=likelihood;`
	`lset nst=1 basefreq=equal;`
	`lset rates=equal pinvar=0;`

F81	`set criterion=likelihood;`
	`lset nst=1 basefreq=empirical;`
	`lset rates=equal pinvar=0;`

K2P	`set criterion=likelihood;`
	`lset nst=2 basefreq=equal tratio=estimate;`
	`lset rates=equal pinvar=0;`

HKY85	`set criterion=likelihood;`
	`lset nst=2 basefreq=empirical variant=hky tratio=estimate;`
	`lset rates=equal pinvar=0;`

F84	`set criterion=likelihood;`
	`lset nst=2 basefreq=empirical variant=f84 tratio=estimate;`
	`lset rates=equal pinvar=0;`

GTR	`set criterion=likelihood;`
	`lset nst=6 basefreq=empirical rmatrix=estimate;`
	`lset rates=equal pinvar=0;`

Using Invariant-sites Model For Rate Heterogeneity

Model	PAUP* commands

JC69+I	`set criterion=likelihood;`
	`lset nst=1 basefreq=equal;`
	`lset rates=equal pinvar=estimate;`

F81+I	`set criterion=likelihood;`
	`lset nst=1 basefreq=empirical;`
	`lset rates=equal pinvar=estimate;`

K2P+I	`set criterion=likelihood;`
	`lset nst=2 basefreq=equal tratio=estimate;`
	`lset rates=equal pinvar=estimate;`

HKY85+I	`set criterion=likelihood;`
	`lset nst=2 basefreq=empirical variant=hky tratio=estimate;`
	`lset rates=equal pinvar=estimate;`

F84+I	`set criterion=likelihood;`
	`lset nst=2 basefreq=empirical variant=f84 tratio=estimate;`
	`lset rates=equal pinvar=estimate;`

GTR+I	`set criterion=likelihood;`
	`lset nst=6 basefreq=empirical rmatrix=estimate;`
	`lset rates=equal pinvar=estimate;`

Using Discrete Gamma Model For Rate Heterogeneity

Model	PAUP* commands

JC69+Γ	`set criterion=likelihood;`
	`lset nst=1 basefreq=equal;`
	`lset rates=gamma ncat=4 shape=estimate pinvar=0;`

F81+Γ	`set criterion=likelihood;`
	`lset nst=1 basefreq=empirical;`
	`lset rates=gamma ncat=4 shape=estimate pinvar=0;`

K2P+Γ	`set criterion=likelihood;`
	`lset nst=2 basefreq=equal tratio=estimate;`
	`lset rates=gamma ncat=4 shape=estimate pinvar=0;`

HKY85+Γ	`set criterion=likelihood;`
	`lset nst=2 basefreq=empirical variant=hky tratio=estimate;`
	`lset rates=gamma ncat=4 shape=estimate pinvar=0;`

F84+Γ	`set criterion=likelihood;`
	`lset nst=2 basefreq=empirical variant=f84 tratio=estimate;`
	`lset rates=gamma ncat=4 shape=estimate pinvar=0;`

GTR+Γ	`set criterion=likelihood;`
	`lset nst=6 basefreq=empirical rmatrix=estimate;`
	`lset rates=gamma ncat=4 shape=estimate pinvar=0;`

Using Invariant-sites and Discrete Gamma Model For Rate Heterogeneity

Model	PAUP* commands

JC69+I+Γ	`set criterion=likelihood;`
	`lset nst=1 basefreq=equal;`
	`lset rates=gamma ncat=4 shape=estimate pinvar=estimate;`

F81+I+Γ	`set criterion=likelihood;`
	`lset nst=1 basefreq=empirical;`
	`lset rates=gamma ncat=4 shape=estimate pinvar=estimate;`

K2P+I+Γ	`set criterion=likelihood;`
	`lset nst=2 basefreq=equal tratio=estimate;`
	`lset rates=gamma ncat=4 shape=estimate pinvar=estimate;`

HKY85+I+Γ	`set criterion=likelihood;`
	`lset nst=2 basefreq=empirical variant=hky tratio=estimate;`
	`lset rates=gamma ncat=4 shape=estimate pinvar=estimate;`

F84+I+Γ	`set criterion=likelihood;`
	`lset nst=2 basefreq=empirical variant=f84 tratio=estimate;`
	`lset rates=gamma ncat=4 shape=estimate pinvar=estimate;`

GTR+I+Γ	`set criterion=likelihood;`
	`lset nst=6 basefreq=empirical rmatrix=estimate;`
	`lset rates=gamma ncat=4 shape=estimate pinvar=estimate;`

Extra topic: Rate variation across classes of sites – the site-specific rates option

Earlier in the lab, we added gamma-distributed among site rate variation to our model of character evolution. This lets the model recognize the fact that (in almost all real datasets) some sites change much more frequently than others. This approach of using a generic, statistical distribution may seem unsatisfying because it does not incorporate much of our biological knowledge or intuition.

Another approach is to define different models of evolution for different classes of characters. The current version of PAUP* does not let you use multiple models simultaneously in a completely flexible way (while MrBayes does). However, PAUP**does* allow one component of the model (the rate of evolution) vary over sites. In PAUP*'s terminology this is the site-specific rates model.

This is an example of a partitioned analysis. The partitioning (the assignment of sites to different rate classes) must be done *a priori*. So, you must tell PAUP* which sites should be assumed to have the same rate of substitution. Once you have decided how to partition the sequences, you *can* ask PAUP* to infer the rates that are assigned to these categories.

One of the most common uses of the site-specific rates option in PAUP* is to let different sites in different codon positions have different rates. This agrees with ample empirical observations and the intuition that because many changes in the third position result in either no change in the amino acid sequence or a conservative change (a similar amino acid). Now we'll walk through an example of using PAUP* to assign different rates according to codon position.

1. Obtain the primate-mtDNA.nex file (it comes with PAUP*), for the duration of the workshop I have posted the file at `http://www.people.ku.edu/~mtholder/sisg/`
2. Open the file in a text editors (for example WordPad) and look at `assumptions` block. This section defines groups of taxa and characters (`TaxSet` s and `CharSet` s in NEXUS lingo). The `UserType` command show how you can define a stepmatrix for parsimony (this command is understood by PAUP*, MacClade, and Mesquite). For the purpose of this lab the important commands are the `CharSet` s that tell us which sites belong to each category: `noncoding, 1stpos, 2ndpos`, and 3rdpos. These `CharSet` names are (as far as PAUP* is concerned) arbitrary names – the name `1stpos` does not tell PAUP* that these are the first codon positions.
3. `Execute` the primate-mtDNA.nex file in PAUP*.
4. To tell PAUP* how to partition the characters, we use the `CharPartition` command (which should be placed in an `Assumptions` block if you want to use the partitioning scheme in more than one PAUP* session). You can define as many ways of partitioning the characters into groups as you like. Each partition is assigned a name. The general syntax is:
   `CharPartition <partition name> = <subset name> : <subset definition> , <subset name> : <subset definition> ;` Where everything in the <> is replaced with an appropriate value. Let's setup a character partition that groups first and second positions in one group, has another group for the third base positions, and a third subset for the non coding DNA sequence. The command is:

   ```
   CharPartition byCateg = fs: 1stpos 2ndpos , t : 3rdpos, n: noncoding ;
   ```

   This creates a partition named `byCateg`, first and second partitions are grouped in the subset `fs`, subset `t` has the 3rd base positions, and noncoding sites go into a category named `n`.

5. It is always a good idea to verify that the program did what you were expecting. We can ask PAUP* to show us how it thinks sites are assigned to subsets. Type the command:

   ```
   ShowCharParts;
   ```

   To see a table of subsets. Each character in the matrix has a column in this table (the table has to be wrapped across multiple lines because it is too long). There will be an `*` each column. This tells you which subset PAUP* thinks this character belongs to.

6. Lets get a tree in memory. A quick parsimony search should work (`BAndB` or `HSearch`).

7. Ask PAUP* to score these trees under maximum likelihood and using the HKY model with no among-site rate heterogeneity. Use the `LSet` command to set up the model:

   ```
   LSet BaseFreq=estimate NST=2 TRatio=estimate Rates = equal PInvar=0
   ```

   and `LScore` commands to get the score. Note the likelihood score.

8. Now add gamma-distributed rate heterogeneity to the model with the command:

   ```
   LSet Rates = gamma Shape = estimate;
   ```

   and rescore the trees. Were the scores significantly better?

9. Now lets see how the site-specific rate model works.

   ```
   LSet rates = siteSpec;
   LScore
   ```

   Ooops! This doesn't work does it? PAUP* does not know how to assign sites to rate categories. We have to tell it to use the `CmdPartition` that we created earlier:

   ```
   LSet rates = siteSpec SiteRates = Partition : byCateg;
   LScore;
   ```

   Note: the word "byCateg" here is the arbitrary name that we assigned to the character partition. Is the likelihood higher than the when we used the site-specific model? Do the the relative rates for each of the categories that PAUP* estimated agree with your intuition of which sites are fast and which are slow?

10. If we have *a priori* hypothesis of relative rates that we'd like to test, then we can tell PAUP* not only what the categories are, we can tell it what rates should be used for each. To do this we need a `RateSet` command. This looks like the `CharPartition` command, but we replace the names of subsets with numbers that determine the relative rates. If we thought that noncoding DNA evolved 2 times faster than third positions, and third positions evolve 2 times faster than first and second positions. Then we can test how well this matches the data by looking at the likelihood ratio between analysis in which we estimate the relative rates (the analysis we just ran) and a constrained analysis in which we specify the rates:

```
RateSet doubling = 1: 1stpos 2ndpos , 2 : 3rdpos, 4: noncoding ;
LSet rates = siteSpec SiteRates = RateSet : doubling;
LScore;
```

We can perform a significance test using 2 times the difference in log likelihoods between these hypotheses as our test statistic, and a critical values from the χ^2-distribution with 2 degrees of freedom (there are 3 categories. We can arbitrarily set one of them to 1.0 and estimate the relative rates for the other 2 categories. Thus in the unconstrained model we estimated 2 more parameters than the constrained version).

11. You can test that PAUP* is doing the right thing, by specifying equal rates for all categories:

    ```
    RateSet eq = 1: 1stpos 2ndpos , 1 : 3rdpos, 1: noncoding ;
    LSet rates = siteSpec SiteRates = RateSet : eq;
    LScore;
    ```

    This should give you the same likelihood as if you tell PAUP* not to use rate heterogeneity (as we did in step 7).

You cannot use the χ^2-distribution with the likelihood ratio test statistic to compare the gamma-distributed rate heterogeneity model and the site-specific rates model (because the models are not nested). I would also suggest that use caution when comparing these models based on their likelihoods. The site-specific rates model usually incorporates more prior biological knowledge (from your definition of partitioning), and often have a better fit. Unfortunately (in PAUP* implementation) you cannot allow for rate variation within your categories. Because some first and second sites (for instance) evolve at a high rate, this constraint may lead to worse behavior than the gamma-distributed form of rate variation. Take a look at:

Buckley, T., C. Simon, and G. K. Chambers, "Exploring Among-Site Rate Variation Models in a Maximum Likelihood Framework Using Empirical Data: Effects of Model Assumptions on Estimates of Topology, Branch Lengths, and Bootstrap Support". *Syst. Biol.* **50**(1):6786, 2001.