Contents

Acknowledgments

Organization

The manual is divided into three sections; the first section provides a general overview of the program as well as descriptions on how to install and configure the program on different operating systems. In addition, this first section provides a ”quick start” tutorial. This hands-on tutorial provides a very brief overview of the basic usage of PAUP*. The tutorial will take you step-by-step through an analysis of one of the sample data files. This tutorial was designed for people with no prior experience using PAUP*. The second section of the manual is a detailed description of using PAUP*. Typically, you will refer to this section to find out what PAUP* can do and get a general idea of how to do it. The last section provides background information concerning parsimony, maximum likelihood, and distance methods. When you need more specific information, you will then refer to this last section.

Typographical and notational conventions

⌘O

⌥L

⇧Z

⇥

⌫

The following typographical conventions are used throughout the manual:

Important term	***Boldface italics*** are used to highlight important terms the first time they are defined.
`user input`	This font is used to represent input supplied by the user, either from a file or the command-line.
PAUP* output	This font is used to identify output generated by PAUP*.
Key	This font is used to represent a key on your keyboard.
command	Command names are shown in boldface type.
Menu Command	Commands that are used from input files or typed on the command line are indicated using all upper-case characters. Commands available for selection from menus are shown in mixed upper and lower case.
Dialog item	This font is used to refer to buttons, checkboxes, and other items contained in dialog boxes or elsewhere on the screen.

In descriptions of data-file and command formats, the following notation is used:

ITEMTEXT	Items typed entirely in uppercase are to be entered as indicated. Input of PAUP* commands is case-insensitive, however, so you may enter command names, keywords, etc., in any combination of upper- and lower-case characters. In addition, PAUP* allows abbreviations of command names and keywords to the shortest unambiguous truncation. Note that other NEXUS-conforming programs (MacClade, in particular; see below) may not accept these abbreviations.
[*Optional-item*]	Brackets around an item means that the item is optional. Square brackets may be nested as in [OptionalItem [AnotherOptionalItem]] In this case, each level of nesting depends on the specification of the item at the next higher level. You should not include these brackets when you enter the command.
{ A | B }	Two or more items enclosed in curly braces and separated by vertical bars indicate a set of mutually exclusive choices. The underlined item (if any) indicates the default setting. You should not include the braces or vertical bar when you enter the command.
variable	Letters, words, and symbols shown entirely in lowercase italics represent variables for which specific information must be supplied by the user when the command is entered.
item... <item1 item2>	An ellipsis indicates that the preceding item may be repeated one or more times. If a group of items may be repeated sequentially, the group is surrounded by angle brackets.
[ ] { } - |<>	These symbols are used to define the command format (see above). Unless otherwise indicated, they should not be typed when the actual command is entered.
; : . , ″ ′ ( )	Other punctuation and symbols should be entered as shown in the command description.

Technical Support

Contacting us Assistance with the use of PAUP* and interpretation of output will be provided only to licensed users. If you are unable to resolve a problem by experimentation or need information that is not available in the User's Manual, you may reach me in any of the following ways:

E-mail:	david.swofford@duke.edu (Internet)
Standard Mail:	David L. Swofford Department of Biological Sciences Duke University
FAX:	(Note that I cannot guarantee a FAXed response)

Use of e-mail is vastly more likely to generate a quick response than the other methods. Please provide the following information in your communication.

1. The exact wording of any error messages that you have received.
2. If a crash occurred, the sequence of events prior to the crash (e.g., commands issued, etc.).
3. The exact version number of your program. You can obtain the version number from the opening screen that is displayed when you first start the program.
4. A copy of your data file.

Discussion Forums

Downloads and Updates

We have set up an FTP server to support PAUP*. We will periodically post updated programs, test versions, documentation files, and other announcements there. To use the FTP server, log in to onyx.si.edu (160.111.64.54) as ″anonymous″ and enter your e-mail address when prompted for a password. The overall structure of the ftp directories is described in a README file in the root directory.

When minor bug-fix releases are issued, we will post an ″updater″ program that will convert any version of PAUP*/Mac 3.1 to the new version. The updater program will only work if you already have a copy of PAUP 3.1.

Moving On

Part I

INTRODUCTION TO PAUP* 4.0

Program Overview

PAUP* version 4.0 is a major upgrade and new release of the software package for inference of evolutionary trees, for use in Macintosh, Windows, UNIX/VMS, or DOS-based formats. The influence of high-speed computer analysis of molecular, morphological and/or behavioral data to infer phylogenetic relationships has expanded well beyond its central role in evolutionary biology, now encompassing applications in areas as diverse as conservation biology, ecology, and forensic studies. The success of previous versions of PAUP: Phylogenetic Analysis Using Parsimony has made it the most widely used software package for the inference of evolutionary trees. In addition, the PAUP manual has proven to be an essential guide, serving as a comprehensive introduction to phylogenetic analysis for beginning researchers, as well as an important reference for experts in the field. With the inclusion of maximum likelihood and distance methods in PAUP* 4.0, the new version represents a great improvement over its predecessors. In addition, the speed of the branch-and-bound algorithm has been enhanced and a number of new features have been added, from agreement subtrees to tests for combinability of data and permutation tests for nonrandomness of data structure. These, along with many other improvements, will make PAUP* 4.0 an even more indispensable tool in comparative biological analysis than were previous editions of the program and manual. PAUP* 4.0 and MacClade 3 use a common data file format (NEXUS), allowing easy interchange of data between the two programs.

1 Installing PAUP*

1.1 Macintosh Version

This chapter describes aspects of installing and running PAUP* that are specific to the Apple®Macintosh™. PAUP* has a standard Macintosh interface including pulldown and pop-up menus, dialog boxes, and scrollable lists. This document assumes that you are familiar with these basic interface elements. A command-line version of PAUP* is also available on Macintosh machines running OS X; however, this version will not be discussed here since an introduction to the command-line version of PAUP* is given in the Portable Version section found later in this chapter (insert ref). Specific analytical features will be discussed in Chapter 2 ”Using PAUP*”.

System Requirements	Macintosh machine with a Core Duo or later processor
	Mac OS X 10.8 or higher

1.1.1 Installation

After installation, the Macintosh version of PAUP* may be started by double-clicking on the PAUP* application or a PAUP* document icon, or by dragging a PAUP* or text document icon onto the PAUP* application icon.

1.1.2 The Main Display Window

The display window

The command line (Windows menu)

The apple menu

Memory Status (Windows menu)

edit >clear display buffer

edit >edit display buffer

controlling the display buffer size (**Options >startup preferences**)

1.1.3 The PAUP Editor*

The PAUP* editor has most of the features expected of a Macintosh text editor, including horizontal and vertical scroll bars, cut, copy, paste, clear, and undo facilities, selection of a "word" by double-clicking, etc. You may also select an entire line by triple-clicking. Most of the commonly used editor commands, such as the cut and paste, can be found under the **Edit** menu or by using the corresponding key binding.

1.1.3.1 Opening the Editor.

There are several different ways to access the PAUP* editor. If you have already executed the data file, the name of the file will appear under the Windows menu. Simply select the file name and the contents of the file will appear in an editor window. If you would like to edit a data file that has not already been executed, choose the **Open** ... item from under the **File** menu. Note, by default, the **Open** dialog box will open automatically when you first start PAUP* (see ref to startup preference for change this option). In the **Open** dialog box, select the edit option so that when the file is opened an editor window will appear with the data file in it. From within the editor, you can then make changes to the data file as you like. Keep in mind however, changes made in the editor will not automatically be applied to subsequent analyses in PAUP*. If you change something in the editor, you must first execute the file (whether or not you saved your editing) before PAUP* can use the changes made in the editor. If you close the window without saving your changes, PAUP* will prompt you for the appropriate action.

If you wish to create a new file, choose the New command under the File menu. An untitled window will appear. You can enter a new data matrix, or paste in data from another file. When you are done, save your work and execute the file, as above.

You can also edit a file in implementations of PAUP* that provide built-in editing capabilities (e.g., portable versions). Use the EDIT command to edit a file using PAUP*'s editor.

Syntax: `Edit` [*file-specification*];

1.1.3.2 Changing the Editor's Settings.

Some basic features of the PAUP* editor can be changed by choosing Options >Editor... menu.

To start, you may change the type font, size, and number of spaces that are equivalent to one tab. Although not required, use of a monospaced font like Monaco or Courier is recommended so that columns of the data matrix will line up cleanly. The default font, PAUPMonaco, is a modification of Monaco that overcomes some of the shortcomings of Monaco. Specifically, it distinguishes between zero and the upper-case "oh," between the lower-case "el" and the upper-case "eye," and increases the size of several punctuation characters for improved readability. Font and tab settings for individual windows can be set by using the **Set Tabs and Fonts** dialog under the **Edit** menu. Font/tab settings chosen there automatically override the default settings. By checking the box **Apply font/tab** changes to all open windows in the Editor options dialog will change the font/tab settings for all open windows.

Clicking the **Auto Indent** box will enable or disable the "autoindent" facilty. If autoindenting is "checked", when you hit return at the end of a line, the cursor is automatically indented to the same column position as the starting point of the previous line.

While the PAUP* editor is not intended to be a full-featured sequence editor, the **Use "sequence editing" cursor mode** is designed to facilitate basic sequence editing operations. In this mode, if an down-arrow (or up-arrow) is hit, the cursor does not necessarily move to the same column in the next (or previous) line. Rather, it goes to the column from which the most recent sequence of identical key presses (which may be a single key) was begun. For example, if you insert two hyphen characters and hit a down-arrow, the cursor will go to the next line but left by two characters, making it easier to insert a series of two-site gaps at the same column location.

1.1.4 Macintosh-Specific Search Options

1.2 Microsoft Windows Version

1.2.1 Installation

1.2.2 The Main Display Window

1.2.3 The PAUP* Editor

1.2.4 Configuring Startup Preferences

1.2.5 Command-line Execution

1.2.6 Shortcuts

Syntax: `DOS` [*dos-command*];

1.3 Portable Version

1.3.1 Installation

1.3.2 The Main Display Window

1.3.3 The PAUP* Editor

1.3.4 Configuring Startup Preferences

1.3.5 Command-line Execution

1.3.6 Shortcuts

2 Your First PAUP* Run

The purpose of this chapter is to get you familiarized with some of the features in PAUP*. As you become more experienced using PAUP*, you will discover that there are many alternative ways to execute the operations described below. For obvious reasons, we have chosen not to describe all the possibilities in this chapter; however, we encourage you to explore other menu and command-line options as your time permits. For this chapter you can use the following dataset:

http://paup.sc.fsu.edu/data/primate-mtDNA.nex

Throughout this chapter we follow several typographical conventions. First, menus, menu items, and items contained in dialog boxes or elsewhere on the screen are given in a bold san serif font. For example, the text **File >Open** means click “File” from the main menu and then select “Open” from the menu items under “File.” Second, text that is intended to be typed by the user at the command-line prompt or into a dialog box is given in a plain fixed-width font. For example, the instructions “Type: `weights 2:1stpos`” mean that everything after “Type:” should be entered exactly as it appears. Finally, interface specific instructions are offset and bulleted, whereas all other text pertains to all of the PAUP* interfaces.

2.1 Setting up your data file: the Nexus format

PAUP* reads your data in a NEXUS file format. All NEXUS files must begin with:

NEXUS file declaration

```
#NEXUS
```

If you open up the file primate-mtDNA.nex you will notice that it is divided into blocks of text, delimited by the words "begin" and "end." The word following "begin" defines the block-type. Notice that the block-type is always followed by a semicolon. In the primate-mtDNA.nex example, the following types of blocks are used: `data`, `assumptions`, and `paup`. There are, however, numerous other NEXUS block-types. In fact, one of the advantages of the NEXUS format is that applications will simply skip over blocks that they do not recognize. Many of these block-types will be discussed in greater detail elsewhere in this book. For a more detailed discussion of the NEXUS format see Maddison et al. (1997).

Between the "begin" and "end" in a block you will find commands that PAUP* will read and execute. Commands are usually followed by several options. Most options will be followed by an "=" sign to define that option. Each option -n PAUP* has a default value, but can be modified by the user. Like a block-type, commands and their associated options are followed by a semicolon in a NEXUS file. At a bare minimum PAUP* requires your Nexus file to have "taxa" and "characters." These are easily defined with the dimensions command in the data block, as shown below.

DATA block:

```
begin data;
    dimensions ntax=12 nchar=898;
    format missing=? gap=- matchchar=. interleave datatype=
        dna;
    options gapmode=missing;
    matrix
```

Here the number of taxa and characters are defined with the dimensions command and the options "ntax" and "nchar", respectively. In this example file, the data block also contains several other commands including the "matrix" command that contains the aligned sequences. If you scroll down the file to the end of the sequences you will see where the data block is terminated with an "end;". Notice that both the "matrix" and "end" are followed by a semicolon.

A user can get information about the options available for any command along with their associated current settings by Typing `command-name ?` in the command-line portion of display window.

Getting help:

- Type ? in the command-line window as in Figure 2.1
- Click **Enter** or hit the Return key

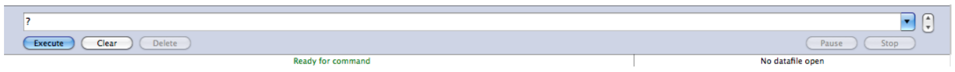


Figure 2.1 *Command-line interface on PAUP* display*

Following the data block you will find a new block named "assumptions." Here you will be able to assign specific assumptions to sets of characters or taxa. These sets allow rapid assignment of character type and weight assumptions. These user defined "Sets" in PAUP* provide a way to refer to collections of objects with single names. The use of sets can greatly reduce the amount of redundant typing needed to issue commands, and can help to avoid mistakes when typing commands into the command line. Each assumption set specifies a character type ("Type Sets"), weight ("Weight Sets") or exclusion status ("Exclusion Sets") for each character. Any number of type, weight, or exclusion sets may be defined. You can then change the assumptions assigned to all characters in the data set simply by invoking a new assumption set.

Before you begin an analysis there is a good chance that you know something about the characters in your data matrix, which might suggest that the characters should be differentially weighted. For example, we know that substitutions at the first codon position generally occur less frequently than substitutions at third positions. The simple explanation for this is that substitutions at first position codons usually result in an amino acid substitution; whereas, third-position changes can occur without changing the amino acid translation. You will incorporate this information into the following analysis by applying a higher weight to substitutions occurring at first position codons. Codon positions have already been identified in the sample file using the charset command.

The use of character sets allows you to refer to a group of characters by a single name, for example a gene that you have sequenced. The only restriction is that the name of the character set cannot be the same as the label assigned to an individual character, for obvious reasons. After defining a character-set using a CharSet command, you can use the character-set name in any place that you would ordinarily use a character name or number. In the example file they are defined like this:

Character Sets ("CHARSETs"):

```
    charset coding = 2-457 660-896;
    charset noncoding = 1 458-659 897-898;
    charset 1stpos = 2-457\3 660-896\3;
    charset 2ndpos = 3-457\3 661-896\3;
    charset 3rdpos = 4-457\3 662-.\3;
```

The CharSet command specifies and names a set of characters; this name can then be used in subsequent CharSet definitions or wherever a *character-list* is required. The name of a CharSet cannot be equivalent to a character name or character number.

By defining a CharSet, it is easy to include and exclude specific sets.

Excluding Character Sets:

- Select **Data >Include/Exclude Characters...**
- Under **Included characters:** box, select the click the **CharSet** pulldown
- Select the **3rdpos** CharSet and click **Exclude**, then click **OK**

Alternatively, this can be done by defining Exclusion sets (EXSETs). Exclusion sets allow you to specify a set of characters that are to be"excluded" from the analysis (see "Excluding Characters" above).

Exclusion Sets ("EXSETs"):

```
exset coding = noncoding;
exset noncoding = coding;
```

By default, PAUP* considers all transformation costs to be equal. In this section, you will invoke a character type that will assign a higher weight to transversions than to transitions. More specifically, we will assume that transversions, changes from a purine (A or G) to pyrimidine (C or T), are two times the cost of transitions, changes from a purine to a purine and pyrimidine to a pyrimidine. One way to incorporate this assumption into the analysis is to set up a transition/transversion "step matrix." Such a step matrix has already been defined in the sample file. To apply the transformation cost to all of the characters currently being considered, do the following:

In addition to defining specific groups of characters in your data, under the assumptions block you can also group specific taxa. The use of taxon sets allows you to refer to a group of taxa by a single name. After defining a taxa-set using a TaxSet command, you can use the taxon-set name in any place that you would ordinarily use a taxon name or number. For example, you could define two taxon sets as follows:

Taxon Sets ("TAXSETs"):

```
taxset hominoids = Homo_sapiens Pan Gorilla Pongo Hylobates;
taxset otherSpp = 1 7-12;
```

and then use the taxon-set names in subsequent commands:

Taxon Sets ("TAXSETs"):

```
delete otherSpp;     [deletes taxa other than those in ‘‘
    hominoids’’]
outgroup otherSpp; [assigns taxa 1 and 7-12 to the outgroup]
constraints ingrp = ((hominoids)); [define a constraint]
```

You cannot declare a TaxSet name that is the same as any taxon name in the data file. But notice that define the set with numbers (8-12) is equivalent to typing out the full taxa names (Macaca_fuscata M._mulatta M._fascicularis M._sylvanus Saimiri_sciureus Tarsius_syrichta).

Use the CharPartition to define a partition of the characters. The CharPartition command is ordinarily issued from within the Sets block. However, you may also issue it from the command line or from within a Paup block. The following example of the CharPartition command creates a character partition named gfunc which defines coding and noncoding regions of the sequences.

Character partitions:

```
charpartition gfunc = 1:2-457 660-896, 2:1 458-659 897-898;
```

The following example of the CharPartition command is equivalent to the previous example except that two predefined character sets are used to define the characters included in each partition.

Character partitions:

```
charset coding = 2-457 660-896;}
charset noncoding = 1 458-659 897-898;

charpartition gfunc = 1:coding, 2:noncoding;
```

Once again, depending on the number of characters in your data matrix, issuing this command can produce a lot of information to be output to the screen. There are no additional options for this command. Notice, however, that in our example file charsets are defined within the assumptions block, whereas the charpartition is defined in a paup block.

2.2 *Executing your data file*

Close the sample file and do the following:

Executing Your Data:

- Double-click the PAUP application icon.
- Select **File >Open...** and select the primate-mtDNA.nex file and click Open
- then Select **File >Execute "primate-mtDNA.nex"**

If you have your Nexus file open in the text editor in PAUP*, you can also execute your file by simply typing ⌘R.

After executing the sample file, PAUP* will display comments and some general information about the data. For this example, the source of the data set is given, followed by a section reporting the dimensions of the data matrix, the type of data, etc.

2.3 Starting a log file

It is a good idea to keep track of things that you are doing in PAUP* by creating a log file. By default, PAUP* will create a log file with the same name as the data file, but with a ".log" suffix on it. However, you can name it anything you want.

Logging output to a file:

- **Select File >Log Output to Disk...**
- Click on **Set...** and **Save as:** practice.log
- If a file names "practice.log" already exists, you will be asked to **Append**, **Cancel**, or **Replace**
- if that is the case, click **Replace**
- Now in the **Log Output** dialog, click **OK**

Logging can be started and stopped anytime during your PAUP* session. To stop logging do the following:

Multiple commands in the command-line:

- Type: `log stop` in the command-line window

Notice that you do not have to end the command with a semicolon (;) in the command-line. However, it is possible to enter multiple commands in the command-line. In such a case you would need to separate the individual commands with a semicolon, like:

Stop logging:

- Type: `log stop; hs; log stop` in the command-line window

2.4 Performing a simple search

PAUP* has the advantage of being able to analyze data using several different optimality criteria; parsimony, likelihood, and distance. Several chapters in this book and a plethora of published literature are devoted to comparing the performance of optimality criteria. Rather than spend time here discussing the relative merits of the available optimality criteria, we will just say that each criterion has its strengths and limitations. To begin with, you will use the default criterion, maximum parsimony, to search for optimal trees. Later in this chapter you will search under the other criteria. For starters, we will search for trees under the parsimony criterion (the default setting in PAUP*)

Defining Optimality Criterion:

- Select **Analysis >Parsimony** (Note: parsimony is the default setting and will probably already be selected).

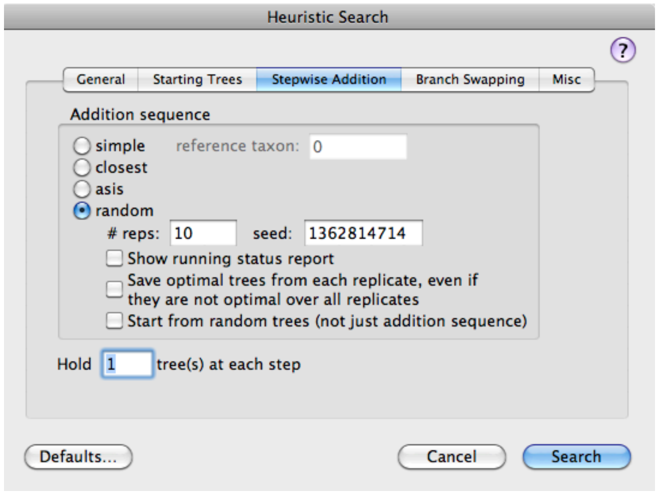
PAUP* provides two basic classes of methods for searching for optimal trees; exact and heuristic. Exact methods guarantee to find the optimal tree(s) but may require prohibitive amounts of computer time for medium to large-sized data sets. Heuristic methods do not guarantee optimality but generally require far less computer time. Even though the current data set is relatively small, you will start by conducting a heuristic search.

Once the search is started, PAUP* will display general information about the options and assumptions being used during the search. If you were logging results, this information would be saved to the log file. When the search completes, PAUP* will display general information about the results of the search.

Performing a Simple Search:

- Select **Analysis >Heuristic Search...**

- Under the options menu chose **Stepwise Addition Options**
- Change **Addition sequence** from **Simple** to **Random** and click **Search**
- Click **Close** to dismiss the search status dialog box


Heuristic Search

General | Starting Trees | Stepwise Addition | Branch Swapping | Misc

Addition sequence

simple reference taxon: 0
closest
asis
[✓ random]
# reps: 10 seed: 1362814714
☐ Show running status report
☐ Save optimal trees from each replicate, even if they are not optimal over all replicates
☐ Start from random trees (not just addition sequence)

Hold 1 tree(s) at each step

Defaults... | Cancel | Search

Figure 2.2 *Heuristic Search Dialog*

It is worth noticing here, that when you execute a command in PAUP*, the equivalent command-line version will be printed to the display buffer.

Command printed to display buffer:

```
paup> HSearch
```

2.5 Viewing trees

According to the output on your screen, there is a single tree currently in memory. To display the tree do the following:

Show trees:

- Select **Trees >Show Trees...**
- Click **OK** and the single most parsimonious tree is printed to the display.

The `showtrees` command draws a simple picture of the branching order of the taxa. Say for example, you want to know something about the branch lengths of the tree. To get a more detailed picture of the tree do the following:

Describe trees:

- Select **Trees >Describe Trees...**
- Under **Output** select **branch-length table**
- Under **Plot type** deselect **cladogram** and select **phylogram**
- Click **Describe**

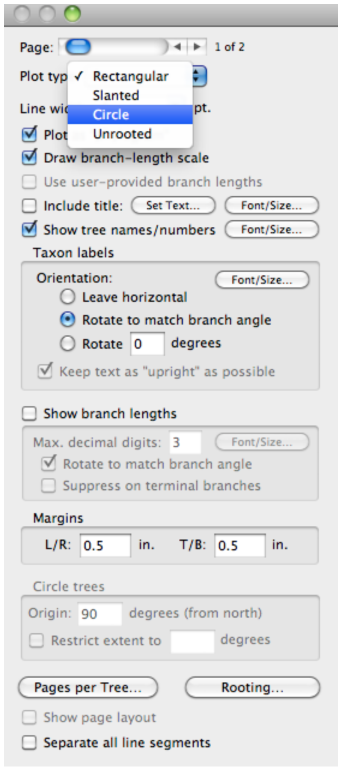
2.6 Printing trees

PAUP* has the ability to print your trees or save them as PDF files to be manipulated with other graphics software.

Printing trees:

- Select **Trees >Print/View Trees...**
- Select **Plot type >CircleTree ...**
- Click the **Show branch lengths** check box and click **Preview**
- Select **Done** and then **Print** if you wish to print the selected tree.

There are many options in the "Print/View Trees..." dialog. Users can change the font, or font sizes, as well as the width of the branches that are being printed.



Page: 1 of 2

Plot typ

Line wi pt.

[✓] Plot

[✓] Draw branch-length scale

[ ] Use user-provided branch lengths

[ ] Include title:

[✓] Show tree names/numbers

Taxon labels

Orientation:

( ) Leave horizontal

(●) Rotate to match branch angle

( ) Rotate 0 degrees

[✓] Keep text as "upright" as possible

[ ] Show branch lengths

Max. decimal digits: 3

[✓] Rotate to match branch angle

[ ] Suppress on terminal branches

Margins

L/R: 0.5 in. T/B: 0.5 in.

Circle trees

Origin: 90 degrees (from north)

[ ] Restrict extent to degrees

[ ] Show page layout

[ ] Separate all line segments

Figure 2.3 *Print/View Trees… Dialog*

2.7 Saving results

More times than not, you will want to save trees to be looked at later. PAUP* can save trees in several different formats: NEXUS, NEXUS (no translation table), Freqpars, Newick (Phylip, Mega, etc...) and Hennig86. To save the tree in NEXUS format:

Saving trees:

- Select **Trees >Save Trees to File...**
- In the **Save Trees as:** dialog box type the file name mp.tre and click **Save**

Like your data file, the newly created trees file is also a NEXUS file (notice the #NEXUS at the beginning of the file). And like other NEXUS files, your tree file stores information in blocks with "begin" and "end." Trees are now stored in a new type of block called the "trees" block. If you look at the .tre file in a text editor, you will also notice that there is a lot of additional information contained in the NEXUS file. Most of this information is contained within square brackets ([]). By convention, PAUP* ignores information contained within these brackets. This allows the user to make notes within the NEXUS file that will not cause problems when the file is executed. There are also cases in the .tre file where there is an "!" following the first bracket ([! text or comment]. When this is placed in a fill, PAUP* will print to the display buffer what is contained in the brackets when the file is executed or read into the program.

2.8 Distance

As mentioned before, PAUP* allows you to search for trees using several optimality criteria; parsimony, likelihood, and distance. PAUP* provides a wide range of pairwise distant measures, from simple absolute differences to more complicated model-based corrected distances. Pairwise distances can be summarized in a table or used to construct UPGMA and neighbor joining trees. In addition, PAUP* can use the minimum evolution and least-squares functions to evaluate trees under the distance criterion. The following section will introduce you to some of these methods.

To change the optimality criterion to distance,

Change optimality criterion to distance:

- Select **Analysis >Distance**

First you will need to choose among the distance measures that PAUP* can calculate. For this example, you can chose the Hasegawa et al. (1985) distance, which estimates a transition/tranversion ratio and base frequencies.

Selecting distance correction:

- Select **Analysis >Distance Settings...**
- In the distance settings dialog box change **DNA/RNA distances** from **Uncorrected ("p")** to **HKY85** and click **OK**
- Select **Data >Show Distance Matrix**

Next, you will construct a neighbor joining tree using the HKY85 distances.

Build a neighbor joining tree:

- Select **Analysis >Neighbor Joining/UPGMA...**
- Click **OK**

Notice here that you could have also gotten to the "Distance Options" dialog by clicking on "Distance options..." in the "Neighbor Joining/UPGMA" dialog box. Also notice that although we selected Distance as our optimality criterion, the "Neighbor Joining" method does not look for an optimal tree. This is also the case if you would have built a UPGMA tree. We can, however, search for tree that evaluate trees under the Distance criterion. Here we can search for trees using the least squares objective function.

Build a least squares tree:

- Select **Analysis >Distance Settings...**
- Under the **Objective function** menu, select **Weighted least squares with standard ("polynomial") weighting**
- **Weighting using power=0** should be the default setting under **Weighted least squares.....** If it is not, then select it now and click **OK**.

- Start the least squares search by selecting **Analysis >Heuristic Search...**
- Click **Search** in the heuristic search dialog box.

2.9 Maximum likelihood

To finish this chapter, you will search for optimal trees using the maximum likelihood (ML) criterion. Under maximum likelihood in PAUP*, an explicit model of nucleotide substitution is used to evaluate trees. Selecting an appropriate model of nucleotide substitution is an important step in a likelihood analysis but is beyond the scope of this chapter. To save time, we have chosen an appropriate model; however, you are encouraged to see Swofford et al. (1996) for a discussion of model selection under the maximum likelihood criterion. Likewise, we touch on this in other parts of this book. Here you will use the parsimony tree, that you saved earlier, to obtain an optimal set of model parameters given the data. Later you will use the same model and set of parameter estimates to search for a maximum likelihood tree.

Set the optimality criterion:

- Select **Analysis >Likelihood**

We have chosen the Hasegawa et al. (1985) model of sequence evolution with gamma distributed rates. Given the parsimony topology and the data we will use PAUP* to estimate the optimal transition/tranversion rate ratio, base frequencies, and among-site rate heterogeniety.

Evaluate the parsimony tree:

- Select **Trees >Get Trees from File...**
- Click **Yes** to dismiss the dialog box warning you that there are unsaved trees.
- Select the file **mp.tre** and click **Get Trees**
- Select **Trees >Tree Scores >Likelihood...**
- In the trees scores dialog box click **Likelihoods settings...**
- In the **Likelihood Settings:** box, select **Substitution Rates**
- Change the **Ti/tv ratio:** from **Set to:** to **Estimate**

- Under the **Maximum likelihood options:** select **Across-Site Rates**
- Under **Across Site Rates** select **Gamma distribution**
- Under **Shape parameter:** change the rate from **Set to:** to **Estimate**
- Click **OK**
- Click **OK** again in the Likelihood Scores dialog box

Depending on the computer you are using it may take a few seconds to several minutes for PAUP* to optimize branch lengths and substitution model parameters on the tree currently in memory. When PAUP* finishes it will output the negative log likelihood of the tree topology found by the parsimony search and give the estimated model parameters values.

Before starting the heuristic search, you will fix the model parameters to those estimated in the previous step. If the options are left to estimate, PAUP* will estimate the parameters on each topology rearrangement made during the heuristic search. Because PAUP* may make thousands of topology rearrangements during a heuristic search, leaving options set to estimate will dramatically increase the time required to complete the search. In general, a more efficient method of estimating model parameters and tree topologies under maximum likelihood is by successively estimating model parameters on novel trees generated by the tree search (Swofford et al., 1996). More specifically, if the topology found under the likelihood criterion differs from that on which the parameters were estimated, then you reestimate parameters on the new topology and search again using the new set of parameters. For this chapter, you will complete one iteration of estimating parameters on a topology and applying the parameters to a subsequent search. In principle, you would continue until you converged on the same topology.

Set likelihood model parameters:

- Select **Analysis >Likelihood Settings...**
- Under the **Likelihood Settings:** select **Substitution Rates**
- Set the **Ti/tv ratio:** to the value estimated in the previous step by clicking the **Previous** button.
- Under the **Likelihood Settings:** select **Across -site rate variation**
- Set the **Shape parameter** to the value estimated in the previous step by clicking the **Previous** button

- Click **OK**

Now you are ready to search under the maximum likelihood criterion. Again, the time required to complete the search will depend on the computer you are using.

Start the tree search:

- Select **Analysis >Heuristic Search...**
- In the **Heuristic Search** dialog box, select the **Stepwise Addition** options
- Under **Addition sequence** change **random** to **asis** and click **Search**

2.10 *Automation: Running PAUP* in batch mode*

Analyses can also be conducted using a non-interactive batch method. This is especially useful when you know your analyses will require a great deal of time to complete, and you don't have time to interact with your computer. You could make elaborate batch files that could analyze multiple datasets under a variety of conditions. In the example below, all the instructions required to complete the sample analyses described above are contained in a "paup" block. A `Set` command was added at the beginning of the paup block to suppress the dialog box indicating that the heuristic search has completed and several other warnings (Mac and Windows only). To run the block in batch mode, copy the text given below to a file and save the file in the same directory as the primate-mtDNA.nex file. Now execute the file as you did the primate-mtDNA.nex file.

PAUP block:

```
Begin paup;
     set autoclose=yes warntree=no warnreset=no;
     log start file=practice.log replace;
     execute primate-mtDNA.nex;
     cstatus;
     include coding/only;
     undelete hominoids lemur_catta
     macaca_fuscata saimiri_sciureus/only;
     weight 2:1stpos;
     ctype 2_1:all;
```

```
          Set criterion=parsimony;
          hsearch addseq=random;
          Showtree;
          describetrees 1/ plot=phylogram brlens=yes;
          savetrees file=mp.tre replace;
          set criterion=distance;
          dset distance=hky85;
          showdist;
          nj;
          dset objective=lsfit power=2;
          hsearch;
          gettrees file=mp.tre;
          set criterion=likelihood;
         lscore nst=2 tratio=est rates=gamma shape=estimate;
          set tratio=previous shape=previous;
          hsearch addseq=asis;
     end;
```

Part II

USING PAUP* 4.0

3
Getting Data Into and Out of PAUP*

Before you can use the analytical features available in PAUP* your data must be stored in a format that can be read by PAUP*. The native format used by PAUP* is called NEXUS, however PAUP* can also import to NEXUS several other commonly used data formats. Table xxx provides a summary of NEXUS data types and other data formats that can be processed by PAUP*.

NEXUS Data Types	Standard DNA RNA Protein
Non-NEXUS Formats	Phylip, Hennig86, TabText Text, GCG, Mega PIR, FreqPars

After the data has been successfully read into PAUP* you can manage the data (e.g., delete or weight characters) and employ the analytical features available in PAUP*. When your analyses are complete, you will likely want to save your results. In addition describing data input procedures, this chapter will also provide a description of what results can be saved by PAUP* and how this can be accomplished.

3.1 NEXUS: the Native PAUP Format

Input files for PAUP* are standard text files containing commands and/or data. For example, input files can be created using any editor that can output text files (i.e., word processors, spreadsheet programs, and text editors). These files will usually adhere to the NEXUS format. The NEXUS format was designed by David Maddison, Wayne Maddison, and David Swofford (Maddison et al., 1997) to facilitate the interchange of input files between programs used in phylogeny and classification. Data files that conform strictly to the NEXUS guidelines can be input to any program that fully supports the NEXUS standards. Currently, these programs include PAUP*, MacClade (Maddison and Maddison, 1992), COMPONENT (Page, 1993), SplitsTree, and Genetic Data Analysis. Among other things, this means that data files created using MacClade's spreadsheet editor can be input to PAUP*, and PAUP tree and data files can be input to MacClade for further analysis. The book accompanying the MacClade program has detailed instructions on the simultaneous use of PAUP* and MacClade.

3.1.1 NEXUS Blocks

The characteristic of NEXUS files that allows them to be so portable is the ”block” concept. A ”block” is a well-defined subsection of an input file that can either be read or ignored by any NEXUS-conforming program. The format of three blocks-Data, Assumptions, and Trees-is defined by the standard; any conforming program must either ignore these blocks entirely or be able to process commands in the block as defined by the standard. [*]

[*]”Process” is defined loosely. Strictly speaking, NEXUS-conforming programs are free to ignore commands within these blocks. For instance, some commands may provide information or instructions that are irrelevant to a particular program. The only requirement is that the program should at least provide the option to continue processing the file after encountering a command that it either does not recognize or does not wish to interpret. (A program can easily skip to the end of the command by looking for its terminating semicolon, even if it is otherwise unable to interpret the command.) The precise behavior of the program upon encountering such commands is not specified by the standard. PAUP* provides the options of allowing execution to continue (with a one-line warning that an unrecognized command was encountered) or terminating the processing of the file (with an error message).

Other blocks are program-specific. If a program encounters a block that it either does not recognize or does not want to deal with, it simply skips over the entire block. (Of course, this feature makes it extremely important to spell block names correctly; otherwise, essential blocks might be unintentionally skipped over during processing). The following is a delicious example of a block that PAUP* will ignore.

```
  #NEXUS
  begin HurricaneRecipe;
   include FreshLemonJuice = 1oz;
   include DarkRum = 4oz;
   include PassionFruitSyrup = 4oz;
   include CrushedIce;
   include OrangeSlice;
   include MaraschinoCherry = 1;
   mix FreshLemonJuice DarkRum PassionFruitSyrup
CrushedIce;
   shake vigorously = 1-2min;
   strain glass = tall;
   garnish MaraschinoCherry OrangeSlice;
  end;
```

For example, the ”taxa” and ”characters” blocks shown below define a simple data set composed of four taxa and five DNA characters.

```
        begin taxa;
            dimensions ntax=4;
            taxlabels
        taxa1
        taxa2
        taxa3
        taxa4
        ;
        end;

        begin characters;
            dimensions nchar=5;
            format datatype=dna;
            matrix
        taxa1   CAACT
        taxa2   GGACT
        taxa3   GGCAC
```

```
taxa4   GGCAC
;
end;
```

Every block starts with "`begin block-type;`" directive and ends with an "`end;`" directive. Each block is composed of one or more commands, each terminated by a semicolon (;). If a command-name within a block is unrecognized, a warning message is issued and the rest of the command ignored by skipping forward to its terminating semicolon. (Optionally, PAUP* will abort processing of a file if an unrecognized command is encountered). Blocks can also be given a name immediately after the block-type but before the semicolon. Currently, PAUP* simply ignores these names.

Within a block, the only restriction on the ordering of commands is the following one: any command which affects the operation of a second command must precede the second command in the file. (i.e., only one pass is made through the file, and all commands are executed immediately when they are encountered; there is no "lookahead" capability).

3.1.2 NEXUS File Identification

NEXUS-conforming files are identified by a `#NEXUS` directive at the very beginning of the file (line 1, column 1).

General format of NEXUS files

NEXUS files are entirely free-format. Blanks, tabs, and newlines may be placed anywhere in the file. Unless RespectCase is requested in the Format command, commands and data may be entered in upper case, lower case, or a mixture of upper and lower case. If RespectCase is requested, case is considered significant in character-state symbols and in names for assumption sets, AncStates specifications, etc. This is only true for standard roman alphabetic characters. For example, `MULLERI` and `mulleri` are not considered identical if Respectcase is off.

Comments may be included in the input file by enclosing them in square brackets. For example,

```
[This is a comment.]
```

Unless the first character following the '[' is an explanation point, the comment is for internal documentation of the data file, and is otherwise completely invisible during the processing of the file. If the comment begins with "[!", as in

```
[!This is a visible comment.]
```

the comment is "visible." NEXUS programs are free to treat visible comments in any appropriate manner; PAUP* simply echoes them to the output destinations (display window, output file, and/or printer) exactly as they appear in the file. Although not a requirement, it is recommended that every NEXUS file begin with a visible comment that at least briefly describes the contents of the file (e.g., source of data, date entered, etc.).

Note to MacClade Users: If you use PAUP* to manually create a NEXUS file for input to MacClade, note that "[!"-style comments are visible only if they are contained within a Data, Taxa, or Trees block.

The following sections provide a brief description of the major elements of the NEXUS format. A more formal description will be published elsewhere (Maddison et al., 1997).

3.1.3 The DATA Block

The Data block contains the data matrix and other associated information. Every data block begins with a Dimensions command specifying the size of the data matrix (number of taxa, number of characters). It is followed by one or more optional commands that specify details of the matrix format, option settings, and character and/or taxon names. The last command in the Data block, Matrix, defines the data matrix itself.

A very simple example of a Data block follows. Note that the style of indenting shown here is not required, but merely serves to enhance readability.

```
begin data;
     dimensions ntax=4 nchar=5;
     matrix
```

```
                  taxon1    00111
                  taxon2    0111?
                  taxon3    11001
                  taxon4    10000
                  ;
         end;
```

Remember that the data matrix, like the rest of the NEXUS input file, is entirely free-format. You may use any number of physical lines to represent each row of the data matrix. We could just have well have entered the matrix above as follows:

```
              matrix
                  taxon1
                  001
                  11
                  taxon2
                  011
                  1?
                  taxon3
                  110
                  01
                  taxon4
                  100
                  00
                  ;
```

or even:

```
              matrix taxon1 00111 taxon2 0111? taxon3 11001
        taxon4 10000;
```

NOTE: In PAUP 2.4, one of the taxa in the data matrix was designated as the hypothetical ancestor. In PAUP 3.1, the definition of the ancestor is no longer part of the matrix, and is done using the AncStates command. This allows more flexibility in changing ancestral states for different analyses of the same data matrix (e.g. to test the effect of alternative polarity assignments).

The full Its syntax is for the Data block is:

```
BEGIN DATA [block-name] ;
     DIMENSIONS NTAX=number-of-taxa NCHAR=number-of-characters;
     [ FORMAT
          [ MISSING=missing-symbol ]
          [ LABELPOS={ LEFT | RIGHT} ]
          [ SYMBOLS="symbols-list" ]
          [ INTERLEAVE ]
          [ MATCHCHAR=match-symbol ]
          [ EQUATE="> symbol = expansion <-]" ]
          [ TRANSPOSE ]
          [ RESPECTCASE ]
          [ DATATYPE = {STANDARD  | DNA  | RNA  | PROTEIN} ]
          [ GAP=gap-symbol ] ; ]
     [ OPTIONS
          [ IGNORE={ NONE  | INVAR  | UNINFORM } ]
          [ MSTAXA = { UNCERTAIN  | POLYMORPH } ]
          [ ZAP = "character-list" ]
          [ GAPMODE = { MISSING  | NEWSTATE} ] ; ]
     [ CHARLABELS character-name -  ; ]
     [ TAXLABELS taxon-name -  ; ]
     [ STATELABELS charnum-and-state-list [ , charnum-and-state-list ] ...
     MATRIX data-matrix ;
END;
```

3.1.3.1 Entering the matrix in "transposed" format

There is considerable flexibility in the form of the input matrix. By default, rows correspond to taxa and columns to characters. If you prefer to list all of the data for each character on a single row, with columns corresponding to taxa, you can use the Transpose option of the Format command. Since the taxon names are no longer defined in the data matrix itself, you can use the TaxLabels command to define the taxon names. For example, the Data block below defines the same matrix as the one above:

```
begin data;
   dimensions ntax=4 nchar=5;
   format transpose;
   taxlabels taxon1 taxon2 taxon3 taxon4;
   matrix
       char1   0011
       char2   0110
       char3   1100
```

```
                char4   1100
                char5   1?10
                ;
        end;
```

3.1.3.2 Placing taxon and character names

Ordinarily, taxon names precede the character-state data for each taxon. If you prefer to place your taxon names at the end of each row rather than t he beginning, you can use the LabelPos = Right option of the Format command (i.e., the names are to the ”right” rather than the ”left” of the data). For example:

```
        begin data;
           dimensions ntax=4 nchar=5;
           format labelpos=right;
           matrix
                00111 taxon1
                01110 taxon2
                11001 taxon3
                10000 taxon4
                ;
        end;
```

You can also use LabelPos = Right in conjunction with the Transpose option. In this case, the character names follow the data for each character.

3.1.3.3 Character-state symbols

You may choose any combination of digits, alphabetic characters, or other symbols to representing character states. The only symbols not allowed are whitespace characters (blank, tab, CR and LF) and the following punctuation characters:

” ’ * ( ) { } [ ] / , ; =

The ”Symbols list” defines the set of symbols that you have chosen to represent character states in the data matrix and in other commands. For data other than molecular sequences (see below), the default Symbols list is ″01″, which means that the only (non-missing) character-state symbols permitted are ′0′ and ′1′. If you want to use any other symbols to designate character states, you must explicitly define an alternate Symbols list in the Format command. The format of a symbols-list is a sequence of single-character ″symbols″ enclosed within double-quotes. For example, if your data matrix contains some two-, three-, and four-state characters, you might specify

```
symbols=”0123”
```

Alternatively, you could specify

```
symbols=”abcd”
```

and use alphabetic characters rather than digits in the data matrix.

To specify a range of digits or alphabetic characters, you can use the ′∼′ (tilde) character. For example,

```
symbols=”0∼9 A∼F”
```

is equivalent to

```
symbols=”0123456789ABCDEF”
```

Note that the tilde is used rather than the hyphen (minus sign) in order that ′+′ (present) and ′-′ (absent) may be used to designate character states (Symbols=″+-″), as in the following example:

```
format symbols=”+-”;
matrix
    Taxon_1- - + -
    Taxon_2- - + -
    Taxon_3+ + - +
    Taxon_4+ - - +
    ;
```

By default, PAUP* does not distinguish between the lower- and upper-case representations of the same alphabetic characters. Thus, if you set the SYMBOLS list to ”abcd”, ’A’ is equivalent to ’a’, ’B’ to ’b’ and so on. If, for some reason, you want to treat the lower- and upper-case representations as different character states, you can use the RespectCase option. For example, the command format respectcase symbols=”ABCDabcd”; defines a Symbols list containing eight distinct character-state designations. The RespectCase option pertains only to symbols used to represent character states in the data matrix. Commands, taxon and character names, etc., can still be entered using any combination of upper- and lower-case characters.

3.1.3.4 Using alphanumeric character names

PAUP* automatically uses the integers 1 through NChar to identify characters for input and output purposes. You may also provide alphanumeric character names to supplement the numeric identifiers using the CharLabels command. For example, the command

```
charlabels amnion appendages thermoreg nostrils teeth;
```

assigns character names to the first five characters. You can then use these character names in any context in which a character number would be used. PAUP* will also use these names to identify characters in the output.

Ordinarily, the CharLabels command will supply a name for each character in the data set. If you provide fewer than NChar names, a warning will be issued (a mistake may have been made in entering the names); only the character numbers will then be available for the missing elements. You may use a single underscore (_) as a placeholder if you want to name some characters but use only numbers for others. For instance, if you wanted to name only the second and fourth characters in the above example, you could use the command

```
charlabels _ appendages _ nostrils _;
```

The remaining characters could then be identified only by number (1, 3, and 5).

You should use a CharLabels command only if the Transpose option is not in effect. If you enter the data in transposed format, the character names will be obtained from the data matrix itself.

See ”Identifiers” for rules pertaining to character names.

3.1.3.5 Predefined formats for molecular sequence data

The predefined formats ”RNA,” ”DNA,” and ”PROTEIN” are available for nucleotide and amino-acid sequence data. These are selected using the DATATYPE option. For example:

```
begin data;
 dimensions ntax=4 nchar=20;
 format datatype=dna gap=-;
 matrix
  TAXON_1  ATGCTATCCGTCATGACCTA
  TAXON_2  ATCCTAGCCGT-AGACGGA
  TAXON_3  CTGCTAGCCGTGGAGTCCTA
  TAXON_4  CTGAA—CGACATAAGTCA
  ;
end;
```

If DataType is set to DNA, the Symbols list is set to ”ACGT” and the following ”equate” macros, corresponding to the IUPAC/IUB ambiguity codes, are predefined:

R =	{AG}	[puRine]
Y =	{CT}	[pYrimidine]
M =	{AC}	[aMino]
K =	{GT}	[Keto]
S =	{CG}	[Strong]
W =	{AT}	[Weak]
H =	{ACT}	[not G]
B =	{CGT}	[not A]
V =	{ACG}	[not T]
D =	{AGT}	[not C]
N =	{ACGT}	[unkNown]

Also, the symbol X is interpreted as ”missing data”.

If DataType=RNA, the Symbols list is set to ″ACGU″ and the same set of equate macros are defined, except that U is substituted for T.

If DataType = Protein, the symbols list is set to ″ACDEFGHIKLMNPQRSTVWY*″, corresponding to the standard IUB single-letter amino acid codes:

A =	ala	[alanine]
C =	cys	[cysteine]
D =	asp	[aspartic acid]
E =	glu	[glutamic acid]
F =	phe	[phenylalanine]
G =	gly	[glycine]
H =	his	[histidine]
I =	ileu	[isoleucine]
K =	lys	[lysine]
L =	leu	[leucine]
M =	met	[methionine]
N =	asn	[asparagine]
P =	pro	[proline]
Q =	gln	[glutamine]
R =	arg	[arginine]
S =	ser	[serine]
T =	thr	[threonine]
V =	val	[valine]
W =	trp	[tryptophan]
Y =	tyr	[tyrosine]
* =	nonsense	[chain termination]

The symbol X, for ″unknown″, is interpreted as ″missing data″. In addition, two ″equate″ macros are predefined:

B = DN [asx = asp or asn]
Z = EQ [glx = glu or gln]

If you want to add additional character-state symbols to the Symbols list implied by the DataType in effect, you can still use the Symbols= option in the format command. These additional symbols will be inserted at the beginning of the Symbols list. For example, if you wanted to mix nucleotide-sequence and restriction-site characters in the same data matrix, you could use the following Format command:

```
format datatype=dna symbols=”01”;
```

The symbols list would then be set to ”01ACGT.” See ”Alignment Gaps” below for another example.

3.1.3.6 Matching the states in the first taxon

You may define character-states relative to the states of the first taxon by using the MatchChar option. For example, the following Matrix command is equivalent to the one above:

```
format datatype=dna gap=- matchchar=.;
matrix
 Taxon_1 ATGCTATCCGTCATGACCTA
 Taxon_2 ..C...G....-A...GG.
 Taxon_3 C.....G....GGA.T....
 Taxon_4 C..AA–..A...A.GTC.
 ;
```

In many cases (particularly with sequence data) it is convenient to separate the data into blocks of characters. This is accomplished using the Interleave option. For the sequence-data example above, we could first supply the data for positions (characters) 1-10 and then for positions 11-20 as follows:

```
begin data;
 dimensions ntax=4 nchar=20;
 format datatype=dna gap=- interleave;
 matrix
  Taxon_1 ATGCTATCCG [1-10]
  Taxon_2 ATCCTAGCCG
  Taxon_3 CTGCTAGCCG
  Taxon_4 CTGAA–CG
  Taxon_1 TCATGACCTA [11-20]
  Taxon_2 T-AGACGGA
  Taxon_3 TGGAGTCCTA
  Taxon_4 ACATAAGTCA
  ;
end;
```

3.1.3.7 Alignment gaps

For molecular sequence data, PAUP* accepts only prealigned sequences. That is, if all of the sequences are not of the same length, ”gaps” corresponding to insertions and/or deletions must be inserted into the sequences at appropriate locations[†]. If your data matrix contains gaps, you must define the symbol used to represent them using the Gap = option of the Format command (see the examples above).

Alignment gaps may be treated either as missing data or as an additional character-state (fifth base or 21st amino acid) using the GapMode option of the Options command. The default is GapMode = Missing. If you want gaps to represent an additional character state, specify GapMode = NewState. You should not use GapMode = NewState when gaps longer than one or two bases occur, because a single evolutionary event (i.e., an insertion or deletion) will then be treated as *n* independent events, where *n* is the length of the gap. If you believe that the gaps convey important phylogenetic information that would be lost by using GapMode = Missing, one strategy is to add an additional set of characters to the data matrix to signify the presence or absence of particular gaps. The Interleave option provides a handy way of adding an additional set of characters to a matrix containing sequence data. For example:

```
begin data;
 dimensions ntax=4 nchar=24;
 format datatype=dna gap=- interleave symbols=”01”;
 options gapmode=missing;
 matrix
  [sequence data...]
  Taxon_1 ATGCTATCCGTCATGA—-
  Taxon_2 ATCCTAGCCGT-AGA—-
  Taxon_3 CT-TAGCCGT-AGTCCTA
  Taxon_4 CT-A—CGA-TAAGTCA

  [insertion/deletion data]
  Taxon_1 0 0 0 1
  Taxon_2 0 0 1 1
  Taxon_3 1 0 1 0
```

[†]The issue of sequence alignment is not a trivial one. Different alignments for the same set of sequences can generate markedly different results. Because current versions of PAUP* do not contain any alignment capabilities, the choice of alignment strategies is left entirely to the user.

```
    Taxon_4 1 1 1 0
    ;
  end;
```

Since the two kinds of characters are in clearly defined blocks, you could easily use other PAUP* commands to assign different weights to insertion/deletion vs. substitution events or to compare trees calculated with and without the information from insertion/deletion characters.

3.1.3.8 EQUATE macros

The Equate macro facility allows translation of particular character-state specifications in the data matrix to alternate character-state specifications. They are useful in several situations. For example, you might use two different symbols to refer to missing data, one for ”unavailable” and another for ”not applicable.” Because PAUP* allows only a single symbol to refer to missing data, you can convert one to other via by equating one symbol to the other. If you, say, issued the command:

```
  format missing=? equate=”- = ?”;
```

then both hyphens and question-marks in the data matrix would be treated as missing data. Equate macros are also useful when more than one character-state is being assigned to some taxa (see ”Multistate Taxa”). For example, if you specify

```
  format equate=”R=AG Y=CT”;
```

then R’s and Y’s (ambiguity codes for purine = A or G and pyrimidine = C or T, respectively) in the data matrix are converted to the corresponding character-state set. In fact, use of DataType = DNA (below) causes this equate macro, as well as others, to be predefined.

3.1.3.9 Using a subset of the characters

By default, PAUP* uses all of the characters in the data matrix. The Ignore and Zap options can be used to restrict the characters actually used by the program. If you request Ignore = Invar, invariant or ”constant” characters are ignored. Alternatively, you can specify Ignore = Uniform to request that ”uninformative” characters be ignored. To ignore specific characters, you can use the Zap option. For example, the command

```
options ignore=uninf zap=”1-10”;
```

causes the first ten characters plus all other uninformative characters to be ignored.

”Ignoring” a character is effectively the same as physically removing it from the data matrix; the only meaningful difference is that physical removal would affect the numbering of the ”downstream” characters, whereas ignoring/zapping it does not. Note that you can also ”exclude” characters; ”ignoring” and ”excluding” are very different operations (see ”Excluding Characters,” in Chapter 2).

An uninformative character is defined as one which contributes exactly the same length to every possible tree topology or, equivalently, one whose minimum possible length is equal to its maximum possible length. For example, an unordered character is informative only if at least two character states each occur in more than one taxon; otherwise, the singleton states can always be explained as single changes on terminal branches, regardless of the tree topology. In general, uninformative characters simply add a constant amount to the tree length and do not otherwise affect the results. However, the above definition of an uninformative character is somewhat ambiguous when missing data are present. In this case, ”uninformative” characters can nonetheless provide potential support for certain groupings, and can therefore affect whether zero-length branches are collapsed (see the sections ”Missing Data” and ”Zero-Length Branches and Polytomies”). Consequently, unless you decline the option to collapse zero-length branches, tree searches may find different numbers of trees depending on the setting of the Ignore option.

Invariant characters are those in which only one non-missing state was observed. Such characters contribute zero length to any possible tree and are therefore also uninformative. When missing data are present, the same ambiguities referred to in the above paragraph with respect to the collapsing of zero-length branches with uninformative characters also apply to ”invariant” characters.

The type of a character can affect its informativeness. For example, a character for which states 0, 1, and 2 were observed once, five times, and once, respectively, would be uninformative as an unordered character but informative as an ordered character.

PAUP* does not attempt to evaluate the consistency index or informativeness of user-defined stepmatrix characters due to the difficulty of determining the minimum possible lengths required by these characters. In principle, this minimum length could be established by performing separate parsimony analyses on each stepmatrix character considered separately, but the amount of computation required would be horrendous. If you absolutely must determine whether a given stepmatrix character is uninformative, you could exclude all other characters and perform an exact (or heuristic) search to obtain (or estimate) the minimum possible length for that character. This minimum length could then be compared to the maximum possible length (available via the Describe command or the **Describe Trees** menu command).

3.1.4 The ASSUMPTIONS Block

The Assumptions block houses assumptions about the data, or gives general directions as to how to treat it (e.g., which characters are to be excluded from consideration). In earlier versions of PAUP, the CharSet and TaxSet commands also appeared in the Assumptions block; now they appear in the Sets block. The GapMode subcommand of the Options command of this block was originally housed in an Options command in the Data block. As this subcommand dictates how data are to be treated, rather than provide details about the data themselves, it was moved into the Assumptions block.

Its syntax is:

```
BEGIN ASSUMPTIONS [block-name] ;
  [ OPTIONS
    [ DEFTYPE=default-character-type ]
    [ POLYTCOUNT={ MINSTEPS | MAXSTEPS } ];
  [ USERTYPE name [ { STEPMATRIX | CSTREE } ]
    = description ; ]
  [ CHARSET character-set-name = character-list ; ]
```

```
 [ TYPESET [*] name = character-type : character-list
   [ , character-type : character-list ] - ; ]
 [ WTSET [*] weight-set-name = character-weight : character-list
   [ , character-weight : character-list ] - ; ]
[ EXSET [*] exclusion-set-name = character-list ; ]
 [ ANCSTATES [*] ancestor-name = character-state : character-list
   [ , character-state : character-list ] - ; ]
END;
```

Available Options

This command houses a number of disparate subcommands. They are all of the form subcommand = option.

DefType = *type-name*

This specifies the default character type for parsimony analyses. Whenever a character's type is not explicitly stated, its type is taken to be the default type. The default DefType is UNORD, but may be changed to one of the following: Ord, Unord, Dollo, Dollo.Up, Dollo.Dn, Irrev, Irrev.Up, or Irrev.Dn.

PolyTCount = { MinSteps | MaxSteps }

This option specifies whether polytomies are treated as "hard" or "soft" (see Maddison (1989)) when counting the number of steps required by a character and when reconstructing ancestral states. PAUP* currently supports only PolyTCount = MaxSteps ("hard" polytomies), and will ignore the PolyTCount specification.

GapMode = { Missing | GapMode | Indel }

Specifies how gaps are to be treated. GapMode = Missing designates gaps to be treated in the same way as missing data; GapMode = GapMode specifies that gaps are to be treated as an additional state (for DNA/RNA data, as a fifth base).

An example Assumptions block follows, with examples of each of the available commands:

```
begin assumptions;
  options deftype=ord polymcount=addsteps;
  usertype myOrd = 4
   0 1 2 3
   - 1 2 3
   1 - 1 2
   2 1 - 1
   3 2 1 - ;
  usertype myTree cstree = ((0,1)a,(2,3)b)c;
  typeset * mixed = irrev: 1 3 10, unord 5-7;
  charset odd = 1-.\2;
  charset even = 2-.\2;
  wtset *one = 2: 1-3 6 11-15, 3: 7 8;
  wtset two = 2:4 9, 3:1-3 5;
  exset nolarval = 1-9;
  ancstates allzero = 0:ALL;
  ancstates allone = 1:ALL;
  ancstates mixed = 0:1 3 5-8 11, 1:2 4 9-15;
end;
```

3.1.5 The TREES Block

The Trees block is used to input user-defined trees to PAUP*. A single Tree or UTree command is used for each tree; any number of TREE or UTREE commands may be included in the block.

The syntax for the Trees block follows:

```
BEGIN TREES [block-name] ;
  [ TRANSLATE token taxon-name [ , token taxon-name ] - ; ]
  [ TREE [*] name = tree-specification; ]
END;
```

Available Options

Translate *token taxon-name* [, *token taxon-name*] - ;]

Use the Translate command to define mappings of arbitrary tokens appearing in the Tree command to valid taxon names. Ordinarily, the tokens are the integers 1 through NTax. The tree description requires references to the taxa defined in a Taxa, Characters, Data, or Distance block.

Tree *name* = *tree-specification*;

Use the Tree command to define trees. If no rooting-specification is supplied, the default in PAUP* is to interpret the tree description as specifying a rooted tree. You can override this behavior by using a ″special comment.″ The comment ′[&U]′ indicates that the tree should be considered unrooted and the comment ′[&R]′ indicates that the tree should be considered rooted. For example:

```
tree mytree = [&R] ((1,2),(3,4));
```

is a rooted tree, whereas

```
tree mytree = [&U] ((1,2),(3,4));
```

is an unrooted tree. You will usually define *unrooted* trees. These trees are then rooted for output purposes using the outgroup or Lundberg rooting procedures. However, if you wish to define an ancestor for the full tree (see AncStates command), if you are using directed characters, or if you want to evaluate trees under a maximum likelihood ″clock″ constraint you may wish to define rooted trees instead. In any case, you can convert rooted trees to unrooted trees and vice versa using the RootTrees/DerootTrees commands.

Note that in order to write the tree descriptions, unrooted trees may be rooted at any convenient point (including a terminal taxon or internal node); the position of the root is simply ignored when the tree is stored.

Another ″special comment″ stored in the Trees block and used by PAUP* is the tree weight comment. Tree weights are the reciprocal of the number of trees found in either a Bootstrap or Jackknife replicate. This option allows the combination of bootstrap results from runs performed at different times or on different machines and the recovery of results obtained prior to a system crash. For example the Treefile created by the Bootstrap command might look like this:

```
[Trees found in bootstrap replicate #1]
tree B_1.1 = [&W 1] (1,((((2,4),5),3),6));

[Trees found in bootstrap replicate #2]
tree B_2.1 = [&W 1/2] (1,((((2,4),5),6),3));
tree B_2.2 = [&W 1/2] (1,(((2,4),(5,6)),3));
```

An example tree with branch lengths is

```
tree PAUP_1 = [&U] (1:22,(2:12,3:9):9,4:72);
```

PAUP* currently ignores the tree names; trees are referred to subsequently only by number, in the order in which they are presented in the Trees block. (MacClade uses the tree names, however.) If the name is preceded by an asterisk (*), the tree becomes the "default" tree. (E.g., a DescribeTrees command with no tree list will result in the description of the default tree.)

If a taxon is omitted from the tree specification, it is assumed to descend from the root of the tree described by the remaining taxa.

Tree descriptions require that taxa be referred to by the name assigned to them in the Data block. However, the Translate command can be used to define a translation table that maps arbitrary tokens in the tree specification to valid taxon names. If a Translate command is not present, a default translation table maps the integers 1 through NTax to the corresponding taxon names in the data matrix, so that integer values rather than taxon names may be used in the tree specifications. However, it is usually best to avoid relying on a default translation table. For example, if you rearrange the order of the taxa in the data matrix, this reordering will not affect the validity of the tree descriptions so long as you either retain the original translation table or use taxon names rather than numbers in the tree descriptions. There are two advantages to using a translation table rather than using the taxon names directly in tree descriptions. First, tree descriptions are generally much more compact when integers rather than taxon labels are used, especially if a large number of trees are being described. Second, if you decide to rename a taxon, only one element of the translation table need be changed. Without a translation table, the old taxon name would have to be changed to the new one in every tree description.

Trees-block processing is much faster if you use the consecutive integers 1, 2, -, NTax for the arbitrary tokens in the translation table. Unless you have a good reason for not doing so, you should follow this convention, especially if a large number of trees are being defined.

A sample Trees block for four taxa named 'one', 'two', 'three', and 'four' is shown below:

```
begin trees;
  translate
   1 one,
   2 two,
   3 three,
   4 four;
  tree a = ((one,two),three,four);
  tree b = ((one,two),(three,four));
  tree c = ((one,two));
  tree d = (1,(2,(3,4))); [uses translation table]
end;
```

If one or more taxa are omitted from a tree specification, they are joined to the root node of the subtree described by that specification. It is recommended that taxa be excluded from tree description only for constraints input. See the section "Defining and using topological constraints."

NOTE: See the section "Manipulating Trees" later in this chapter for more information on defining and using trees, and for a description of the syntax of the Translate, Tree, and Utree commands.

3.1.6 TAXA and CHARACTERS Blocks

An alternative to the use of the Data block is the combination of Taxa and Characters blocks. The Taxa block contains only a Dimensions command and a TaxLabels command. Its Dimensions command specifies only the number of taxa, as Ntax = *n*. The Characters block is essentially identical to the Data block, except that the Dimensions command specifies only NChar = *c* and no TaxLabels command may be present. Thus, the information contained in Taxa and Characters blocks is exactly complementary.

For example, the data in the Data block

```
begin data;
 dimensions ntax=4 nchar=5;
 matrix
  taxon1 00111
  taxon2 0111?
  taxon3 11001
  taxon4 10000
 ;
end;
```

could be defined equivalently as:

```
begin taxa;
 dimensions ntax=4;
 taxlabels taxon1 taxon2 taxon3 taxon4;
end;
begin characters;
 dimensions nchar=5;
 matrix
  taxon1 00111
  taxon2 0111?
  taxon3 11001
  taxon4 10000
 ;
end;
```

If you want to do analyses that only require information about taxa (e.g., user-defined tree input and consensus tree calculation), you may omit the Characters block entirely. Alternatively, you can include one Taxa block and several Characters and Paup blocks (see below) to analyze different data matrices for the same set of taxa. Note that taxon labels in the characters block are just placeholders; the taxon labels defined in the Taxa block are used for the remainder of the data file.

PAUP* supports the Taxa block mainly for compatibility with Rod Page's COMPONENT program (Page, 1993), the latest version of which uses the NEXUS format.

3.1.7 The TAXA Block

The Taxa block specifies information about taxa. In previous versions of PAUP, both the taxa information and the character matrix were contained in the Data block. However, in some instances the character matrix is not needed, making the option to input only taxa labels useful. While it is still permitted, we strongly recommend against use of a Data block or definition of taxa in a Characters or Distances block because this will prevent some programs from extracting taxon information from the file.

```
BEGIN TAXA;
  DIMENSIONS NTAX=number-of-taxa;
  TAXLABELS taxon-name taxon-name ...;
END;
```

Available Options

Dimensions must appear before TaxLabels, and only one of each command is allowed per block.

Dimensions NTax= *number-of-taxa*;

where *number-of-taxa* and are integer values.

TaxLabels *taxon-name taxon-name* ...;

The TaxLabels command defines taxa, specifies their names, and determines their order.

See ”Taxon identifiers” for rules on the format of taxon names.

Taxa may also be defined in the Characters and Distances blocks, if the Newtaxa token is included in the Dimensions command; see the descriptions of those blocks for details.

3.1.8 The CHARACTERS Block

A Characters block contains the data matrix and other associated information. Taxa are usually not defined in a Characters block; if they are not defined here, this block must be preceded by a block which defines taxon labels and ordering (e.g., Taxa). The syntax is:

```
   BEGIN CHARACTERS;
     DIMENTIONS
       [ NEWTAXA NTAX=number-of-taxa ]
       NCHAR = number-of-characters;
     [ FORMAT
       [ DATATYPE = { STANDARD | DNA | RNA | NUCLEOTIDE |
PROTEIN} ]
       [ RESPECTCASE ]
       [ MISSING = symbol ]
       [ GAP = symbol ]
       [ SYMBOLS = "symbol symbol..." ]
       [ EQUATE = "symbol=entry [ symbol=entry... ]" ]
       [ MATCHCHAR = symbol ]
       [ LABELS = { NO | LEFT | RIGHT} ]
       [ TRANSPOSE ]
       [ INTERLEAVE ]
       [ [NO]TOKENS ]; ]
     [ ELIMINATE character-set; ]
     [ TAXLABELS taxon-name taxon-name...; ]
     [ CHARSTATELABELS
       character-number [ character-name ]
       [ /state-name [ state-name... ] ], ...; ]
     [ CHARLABELS character-name [character-name... ]; ]
     [ STATELABELS character-number [ state-name [ state-name... ] ],...; ]
     MATRIX data-matrix;
   END;
```

Dimensions, Format, and Eliminate must all precede Charlabels, Statelabels, and Matrix. Dimensions must precede Eliminate. Only one of each command is allowed per block.

Dimensions

The Dimensions command specifies the size of the data matrix. Ordinarily, the Characters block is proceeded by a Taxa block which contains taxa information. However, if new taxa are to be defined in the Characters block, this must be indicated by the NewTaxa subcommand. NewTaxa, if present, must appear before the NTax subcommand. The number following NTax is the number of characters in the data matrix.

Format

The Format command is used to specify information pertaining to the format of the data file. The DataType subcommand must appear first in the command. The Respectcase subcommand must appear before the Missing, Gap, Symbols, and MatchCase subcommands.

Any or all of the following option specifications may be given:

DataType = { Standard | DNA | RNA | Nucleotide | Protein }

This subcommand specifies the class of data. If present, it must be the first subcommand in the FORMAT command. If DataType = Standard, the Symbols list is taken from the Symbols = ”`symbols-list`” item, above (default = ”01”). If DataType is set to one of the molecular sequence types, a predefined Symbols list is used (”ACGT” for DNA, ”ACTU” for RNA, and the standard one-letter amino acid codes for PROTEIN). In addition, standard ambiguity codes are implemented by predefined Equate macros.

RespectCase

By default, PAUP* does not distinguish between upper- and lower-case character-state symbols in the data matrix. If you want upper- and lower-case representations of the same alphabetic character to refer to different character states, specify RespectCase. This subcommand must appear before the Symbols subcommand and is not applicable to Datatype = DNA, RNA, Nucleotide, and Protein.

Missing = *symbol*

The *symbol* specifies a character used to represent missing data. Any alphabetic, numeric, or other character that may be used as a character-state symbol may be used as the *missing-symbol*. If Missing is not specified, it defaults to ′?′. Whitespace is illegal as a missing data symbol, as are the following:

( ) [] {} / ; := * ′ ″‘ <>̂

Gap = *symbol*

The *symbol* a character used to represent alignment gaps, corresponding to insertions and/or deletions. For example, Gap = - would assign the hyphen as the gap character. The Gap setting is ignored unless Datatype is DNA, RNA, or Protein.

Alignment gaps may be treated either as missing data or as an additional character-state (fifth base or 21st amino acid) using the Gapmode option. Whitespace is illegal as a gap symbol, as are the following:

( ) [] {} / ; := * ′ ″‘ <>̂

Symbols = ″*symbol symbol...*″

The *symbol* list defines a set of permissible symbols that may be used to designate character states. The default Symbols list is ″01″ for the Standard data type (see below), which means that the only (non-missing) character-state symbols permitted are ′0′ and ′1′. If you want to use any other symbols to designate character states, you must explicitly define an alternate Symbols list.

The format of a *symbols-list* is a sequence of single-character ″symbols″; the entire list is then enclosed within double-quotes. Whitespace is not needed between elements: Symbols = ″012″ is equivalent to Symbols = ″0 1 2″.

Equate = ″*symbol = entry*″

The Equate option provides a simple macro facility for translating character-state specifications in the data matrix to alternate character-state specifications. The symbol component must be a single-character. *Entry* is either a valid character-state or a character-state set .

Any number of Equate macros may be specified following the equal sign, but only one pair of double-quotes is used. For example:

```
        format equate=”U=T R=AG .=- X=?”;
```

Equate macros may not be defined recursively. That is, you cannot equate A to B and B to C, expecting A to be expanded to C. If you equate the same symbol to more than one expansion, the last definition applies. Case is significant in equate symbols. That is, Missing = ? Equate =”E=(012) e = ?” means that E will be interpreted as 0, 1, and 2, e as missing data.

MatchChar = *symbol*

If a *symbol* is specified, any occurrence of that symbol in the data matrix is translated to the state (or state-set) occurring in the first row of the matrix. For example:

```
format datatype=dna gap=- matchchar=.;
 matrix
 One     ATGCTATCCGTCATGACCTA
 Two     ..C...G....-A...GG.
 Three   C.....G....GGA.T....
 Four    C..AA—..A...A.GTC.
 ;
```

Whitespace is illegal as a matching character symbol, as are the following:

() [] , ; = * ’ ”‘ / <>̂

Labels = No | Left | Right

If Labels = Left, taxon names (or character names, if the data matrix is transposed) begin each row of the data matrix (i.e., precede the character-state data). If Labels = Right, then these names end each row of the data matrix. The default is Labels = Left. The Left and Right options are PAUP* extensions. We recommend the use of Labels or Nolabels for compatibility with other NEXUS programs. Labels with no other qualifier is equivalent to Labels = Left.

Example:

```
begin data;
 dimensions ntax=4 nchar=5;
 format label = right;
 matrix
  00111 taxon1
  01111 taxon2
  11001 taxon3
  10000 taxon4
   ;
end;
```

Transpose

If Transpose is specified, rows of the data matrix correspond to characters and columns correspond to taxa. Otherwise, rows correspond to taxa and columns to characters. The following is an example of a transposed data matrix:

```
begin data;
  dimensions ntax=4 nchar=5;
  format transpose;
  taxlabels taxon1 taxon2 taxon3 taxon4;
  matrix
   char1 0011
   char2 0110
   char3 1100
   char4 1100
   char5 1010
    ;
  end;
```

Interleave

Specification of allows the data to be entered in ”blocks” of characters. If the data is not transposed, then each ”block” contains information for some of the characters, for all taxa. Taxa in each section must occur in the same order. This format is especially useful for molecular sequence data, where the number of characters can be large. A small interleaved matrix follows:

```
matrix
   taxon1 A C C T C G G C
   taxon2 A C C T C G G C
   taxon3 A C G T C G C T
   taxon4 A C G T C G C T

   taxon1 T T A A C G A
   taxon2 T T A A C C A
   taxon3 C T C A C C A
   taxon4 T T C A C C A
   ;
```

The interleaved sections need not all be of the same length. In an interleaved matrix, new line characters are significant; they indicate that the next character information encountered applies to a different taxon (for non-transposed matrices).

[No]Tokens

The Tokens subcommand is currently not supported by PAUP* 4.0; however, this subcommand is described here because it will be supported in the final release 4.0. This subcommand specifies whether or not data matrix entries are single symbols, or can be tokens. If Tokens is specified, then the data values must be full NEXUS tokens, separated by whitespace or punctuation as appropriate, as in the following example:

```
begin characters;
 dimensions nchar=3;
 format tokens;
 matrix
  taxon_1 absent red big
  taxon_2 absent blue small
  taxon_3 present blue small;
end;
```

[No]Tokens is the default and is not allowed for Datatype DNA, RNA, and Nucleotide. If Tokens is invoked, the standard 3-letter amino acid abbreviations can be used with Datatype = Protein, and defined state names for Datatype = Standard.

Eliminate

This command allows one to specify a list of characters that are to be excluded from consideration. PAUP* will completely ignore eliminated characters. (This is similar to the Zaps subcommand in version 3.1 of PAUP.)

Example:

```
  eliminate 4-100;
```

tells the program to skip over characters 4 through 100 when the matrix is read. Character-set names are not allowed in the character list. This command does not affect character numbers.

TaxLabels

This command allows one to specify the names of the taxa. It serves to define taxa, and is only allowed in a Characters block if the Newtaxa token is included in the Dimensions statement.

CharStateLabels

CharStatLabels allows one to specify both the names of the characters and names of the states. This command was developed as an alternative to the older commands CharLabels and StateLabels.

Example:

```
   charstatelabels
    1 eye_color/red blue green,
    3 head_shape/round square,
    5 pronotum_size/small medium large
   ;
```

A forward slash (/) separates the character name and the state names, with a comma separating the information for different characters. If no state names are to be specified, the slash may be omitted; if no character names are to be specified, the slash must be included, but no token needs to be included between the character number and the slash. If state *x* is the last state to be named, then subsequent states need not be named but states 1 through *x* must be. If no name is to be applied to a state, enter a single underscore for its name. Character names must not correspond to another character name or number; thus, 1 is not a valid name for the second character listed. PAUP* does not use character-state label information; however, it will extract the name from a CharStateLabels command.

CharLabels

This command allows one to specify names of characters:

```
charlabels
  flange microsculpture body_length
  hind_angles #_spines spine_size
  _ _ head_size pubescent_intervals
  head_color clypeal_margin;
```

Character labels are listed consecutively. If character *x* is the last character to be named, then subsequent characters need not be named, but characters 1 through *x* need to be. If no name is to be applied to a character, a single underscore can be used for its name. Character names must not correspond to another character name or number; thus, 1 is not a valid name for the second character listed.

The command should be used only for non-transposed matrices (in transposed matrices, the character labels are defined in the Matrix command).

StateLabels

Use this command to assign names to character states. PAUP* does not use this information in any way, however, the information is used by MacClade and possibly other NEXUS programs.

```
statelabels
1 absent present,
2 isodiametric transverse,
3 '4.5-6.2mm' '6.3-7.0mm' '7.7-11.0mm',
4 rounded subangulate angulate,
10 0 '1-4' '6-9' '7-9' '8-9' 7 8 9,
11 black rufous metallic flavous,
12 straight concave,
;
```

State labels need not be specified for all characters. A comma must separate state labels for each character. State labels are listed consecutively within a character. If state *x* is the last state to be named, then subsequent states need not be named, but states 1 through *x* must be. If no name is to be applied to a state, enter a single underscore for its name.

Matrix

In its standard format, the Matrix command contains a sequence of taxon names and state information for that taxon. The matrix itself may appear in a variety of forms.

Example:

```
begin data;
 dimensions nchar=5;
 matrix
  taxon1 00111
  taxon2 01110
  taxon3 11001
  taxon4 10000
  ;
end;

begin data;
 dimensions nchar=5;
 matrix
  taxon1
  001
  11
  taxon2
  011
  10
  taxon3
  110
  01
```

```
     taxon4
     100
     00
     end;


   begin data;
    dimensions nchar=5;
     matrix taxon1 00111 taxon2 0111? taxon3 11001
     taxon4 10000;
     end;
```

Taxa need not be in same order as in the Taxa block, and the matrix need not contain all taxa. For interleaved matrices, all sections must have the same taxa in the same order.

3.1.9 *The DISTANCES Block*

This block contains distance matrices. Taxa are usually not defined in a Distances block; rather, this block should be preceded by a block that defines taxon labels and ordering (e.g., Taxa). The syntax of the block is as follows:

```
Begin Distances;
     [ Dimensions [ NewTaxa ] NTax = number-of-taxa
          NChar = number-of-characters;]
     [Format
           [Triangle = {Lower  | Upper | Both} ]
          [ [No] Diagonal]
          [ [No] Labels]
           [Missing = Symbol]
          [Interleave] ; ]
     [ TaxLabels taxon-name taxon-name...;]
     Matrix distance-matrix;
End;
```

Commands must appear in the order listed. Only one of each command is allowed per block.

3.1.9.1 Dimensions

The Ntax subcommand of this command is needed to process the matrix when some defined taxa are omitted from the distance matrix. The Nchar subcommand is optional, and can be used to indicate the number of characters for those analyses that need to know how many characters (if any) where used to calculate the distances. Nchar is not required for successful reading of the matrix.

As for the Character block, taxa can be defined in a Distances block if NewTaxa precedes the NTaxa subcommand in the Dimensions command. It is advised that new taxa not be defined in a Distances block, for the reasons discussed in the description of the Data block. NewTaxa, if present, must be appear before the Ntaxa subcommand.

3.1.9.2 Format

This command specifies the formatting of the Matrix. The [NO]Labels and Missing subcommands are as described in the Characters block.

Triangle = {Lower | Upper | Both }

This subcommand specifies whether only the lower left half of the matrix is present, or only the upper right, or both halves. Below is one example of an upper triangular matrix and one of a matrix with both halves included.

```
begin distances;
     format triangle = upper;
     matrix
          taxon_1   0.0     1.0     2.0     4.0     7.0
          taxon_2           0.0     3.0     5.0     8.0
          taxon_3                   0.0     6.0     9.0
          taxon_4                           0.0    10.0
          taxon_5                                   0.0;
end;
begin distances;
     format triangle = both;
     matrix
```

```
          taxon_1   0        1.0      2.0      4.0      7.0
          taxon_2   1.0      0        3.0      5.0      8.0
          taxon_3   2.0      3.0      0        6.0      9.0
          taxon_4   4.0      5.0      6.0      0       10.0
          taxon_5   7.0      8.0      9.0      10.0     0;
end;
```

3.1.9.3 Diagonal

If Diagonal is turned off, the diagonal elements are not included

```
format nodiagonal;
matrix
 taxon_1
 taxon_2 1.0
 taxon_3 2.0 3.0
 taxon_4 4.0 5.0 6.0
 taxon_5 7.0 8.0 9.0 10.0;
```

If Triangle = Both, then there will be one row that contains only the name of a taxon. This row is required. If Triangle = Both, then the diagonal must be included.

3.1.9.4 Interleave

The Interleave subcommand in the Distance block is currently not supported by PAUP* 4.0 beta; however, this subcommand is described here because it will be supported in the final release 4.0. The Interleave format is similar to that described in the Characters block, although the interleaved distance matrices take a slightly different form:

```
taxon_1     0
taxon_2     1     0
taxon_3     2     3     0
taxon_4     4     5     6
taxon_5     7     8     9
taxon_6     11     12     13

taxon_4     0
```

```
     taxon_5     10     0
     taxon_6     14     15     0;
```

As in the Characters block, new line characters in interleaved matrices are significant, in that they indicate a switch to a new taxon.

3.1.9.5 TaxLabels

Taxlabels allows one to specify the names and ordering of the taxa. This command serves to define taxa, and is allowed only if the Newtaxa token is included in the Dimensions statement.

3.1.9.6 Matrix

The Matrix command contains the distance data.

3.1.10 The PAUP Block

One or more Paup blocks may be included in the data file in addition to the standard blocks. Any valid Paup command may be placed in these blocks (in fact, any command outside of NEXUS blocks is assumed to be a Paup command). A typical Paup block would look like:

```
begin paup;
 log file=my.output replace;
end;
begin data;
 .....
end;
begin paup;
 ctype ord: all;
 bandb;
 savetrees file=my.trees;
end;
```

Note: See "Commands Used in the PAUP Block or from the Command-Line" for a description of all PAUP commands.

In many cases, it may be easier to include commands in the file than to enter them from the command line or to make the corresponding requests via the menu system. An example follows:

```
#NEXUS
begin paup;
 exclude 1 5 10-12;
 wts 2:1-10, 3:11-20;
 set maxtrees=200;
 hsearch/keep=120;
 describe all/chglist apolist;
end;
```

The advantage of keeping separate command file is that the data matrix need only be executed a single time. When each command set is needed, simply open it and execute it. You may also use the same command file on different data sets, providing the number of characters and/or taxa do not conflict.

3.2 Importing data

Use the ToNexus command to convert a file to NEXUS format.

Command-line: `ToNEXUS` [*options*];

Menu equivalent: **File >Import Data** ...

Available Options:

- **Format = Phylip|Hennig86|TabText|Text|GCG|Mega|PIR|FreqPars**
- **FromFile =** ***foreign-file-name***
- **ToFile =** ***NEXUS-file-name***
- **DataType = Nucleotide|Protein|RestSite|Standard|Distance |DNA**
- **Interleave = Yes|No**

- **GapSymbol** = ***single-character-value***
- **MissSymbol** = ***single-character-value***
- **IdentSymbol** = ***single-character-value***
- ***Replace** = **Yes|No**

*Option is nonpersistent

Description of options

Format = Phylip|Hennig86|TabText|Text|GCG|Mega|PIR | FreqPars

Use Format to specify the import file-type. You may choose from the following formats:

1. PHYLIP (Felsenstein)
2. Hennig68 (Farris)
3. GCG MSF
4. Mega (Kumar, Tamura, and Nei)
5. NBRF-PIR
6. Text (tab or space delimited)
7. FreqPars (Swofford and Berlocher)

FromFile = foreign-file-name

Use Fromfile to specify the foreign-file-name to be imported.

ToFile = NEXUS-file-name

Use Tofile to specify the NEXUS-file-name used to save the NEXUS-formatted data set.

DataType = Nucleotide|Protein|RestSite|Standard| Distance|DNA

If Format = Phylip|Tabtext|Text, then use Datatype to specify the type of data being imported. If Format = Mega, then Datatype =

Nucleotide|Protein|Distance may be selected. Note that PAUP* currently does not support distance formats.

Interleave = Yes|No

Use Interleave if Format = Phylip and the data set in the PHYLIP file are interleaved.

GapSymbol = *single-character-value*

If Format = Tabtext|Text|Mega, then a single-character-value may be entered to represent gaps.

MissSymbol = *single-character-value*

If Format = Tabtext|Text|Mega, then a single-character-value may be entered to represent missing data.

IdentSymbol = single-character-value

If FORMAT = TABTEXT|TEXT|MEGA, then a single-character-value may be entered to represent an identity character. This is equivalent to what PAUP* refers to as MATCHCHAR.

Replace = Yes|No

If the NEXUS-file-name used under the Tofile option already exists you will be prompted for confirmation that the existing file should be replaced. Replace suppresses this prompt; the existing file will be quietly overwritten by the new data.

3.3 Executing your data file

Use the Execute command to request processing of an input file. The input file should be a valid NEXUS file.

Syntax: : `Execute` *file-specification*;

See "Input/Output files" for details regarding the format of file-specification

The input file may contain any or all of the following: Taxa blocks, Characters blocks, Data blocks, Assumptions blocks, Trees blocks, Distance blocks, Codons blocks, and valid Paup commands. Although not required, you should place PAUP commands inside of a Paup block so that other programs (e.g., MacClade) can use the same file. PAUP commands are processed exactly as if they had been entered from the command line, with only a few exceptions (e.g., the Edit command cannot be issued from a file). Commands are processed until the end of the file is reached, or a Quit command is encountered.

Menu equivalent: **File >Execute "your-data-file"**

3.4 Editing your data file

Use the Edit command to edit a file using PAUP*'s editor.

Syntax: : `Edit` [*file-specification*];

This command is only valid in implementations of PAUP* that provide built-in editing capabilities.

3.5 Batch Processing

If you wish to run (unattended) a single batch file with multiple data sets and analyses, you must use two Set options in the first Paup block (Nowarnreset and Autoclose). This is because by default Paup will display a warning if you try and reset the data file, and will not close windows unless told to do so, thus you would never proceed beyond the first data set. Batch files allow unattended multiple analyses-useful if you have many data sets to run but do not wish to (or cannot) be present. The format of a typical file with multiple data sets would be:

```
#NEXUS
begin data;
  ......; [first data set]
end;
begin paup;
  set nowarnreset autoclose;
  outgroup 1 2 3;
  hsearch swap=tbr addseq=simple hold=10;
end;


begin data;
  ......;[second data matrix here]
end;
begin paup;
  ..... ;[paup commands - new search]
end;
```

Each analysis may have its own Trees or Assumptions block, as these are reset when a new data set is processed.

3.6 Error Messages and Input Files

Because of the extremely free format of data-entry, it is sometimes hard for PAUP* to figure out exactly where a user erred when it tries to execute an input file. This leads to ″bogus″ error messages, in that what PAUP* reports is not the real error, but a reflection of a different error earlier in the file. This is a function of the way PAUP* files are constructed. The alternative is to require more rigid formatting, which is acceptable neither to the programmer nor to users. Below are some examples of more common errors.

The number of taxa and/or characters often get changed as an analysis progresses, but if the Nchar or Ntax settings do not match the matrix, PAUP* will complain in an indirect way. For example, the following matrix has seven characters, but setting Nchar to 6 leads PAUP* to complain about the ″t″ in taxon B in the following way ″Invalid character state ′t′ for taxon 2, character 1.″ This means that PAUP* was expecting a character here, not an alphanumeric taxon name. PAUP* did as it was told-it read the first six characters of taxon 1, took the next ″1″ to be the label for taxon 2, the carriage return to indicate that data was coming, and the following ”t” to be the first character state, about which it complained. So the error is really three lines up in the file, not where PAUP* places its flag.

```
begin data;
 dimensions ntax=5 nchar=7;
 format missing=? ;
 matrix
 taxonA 0000001
 taxonB 0100000
 taxonC 1110000
 taxonD 1111101
 taxonE 1101100
 ;
end;
```

If you mistakenly set Ntax to four when it should be five, PAUP* will return the error ”expecting semicolon at end of data matrix.” This tells you immediately that PAUP* had processed all the taxa you told it to, and was not expecting any more. Note that this is the same message you would get if you actually did omit the terminal semicolon from matrix.

This brings up an important point: PAUP*'s free format works only when the semicolons are correctly placed, so that it knows when to start and stop reading a particular part of the input file. For example, omitting the semicolon at the end of the following ”hsearch” command in the Paup block would return the error message ”keyword 'set' not recognized.”

```
hsearch swap=nni hold=10 addseq=asis
set tcompress;
```

PAUP* thought that the Set command was part of the preceding Hsearch command, which lacked a terminating semicolon. You should be tipped off that PAUP* thought Set was a keyword when you know it is a command. There is nothing wrong with the Set command, only with the preceding Hsearch. Adding the missing semicolon will cause both commands to be executed correctly.

Again, PAUP* will do its best to flag errors in an input file, but if you do get a seemingly nonsensical error message, the best strategy is to work backwards in the file to see where that message might be appropriate. If you read carefully what PAUP* thought it should find and think about what might have misled it earlier, chances are you won't have to go far to find the cause. If you use MacClade, you can largely avoid these types of errors by using MacClade's built-in spreadsheet editor, which insulates you from having to understand the details of the NEXUS format.

3.7 Importing and Exporting Data and Trees

3.7.1 Input/Output files

A number of PAUP* commands contain options to input or output data. Unless PAUP* is told otherwise, the default path-name for an input/output file is the path-name of the most recently accessed file. To override this default you must specify an explicit path-name. Path-names must follow operating system conventions. For example:

OS	Syntax
Macintosh	Harddrivename:Tree Folder:treefile.trees
Dos	C: \Tree_Fol\treefile.tre
Window95/NT	C:\Tree Folder\treefile.trees
UNIX	/home/users/Bob/tree_folder/treefile.trees

Unlike the PAUP* convention for identifiers, underscores (_) are not translated into blanks. File-names containing blanks must be enclosed within single-quotes for the Macintosh and Window95/NT releases and are not allowed under either the DOS or Unix releases. See your operating system documentation for other file naming conventions.

3.7.2 Exporting data

PAUP* has the capability of exporting trees and data in the format of other programs, saving you the effort of editing NEXUS files yourself. This gives you the option of importing your trees and data matrix into another program for comparative analysis, and vice versa. Files can be exported in Hennig86 or PHYLIP formats, and you can choose to export the data matrix, the trees, or both. Exported files can then be read directly by the other program. You also have the option of exporting the file as tab-delimited or plain text, should you wish to import the file into a spreadsheet or word-processing program. The only things that are preserved when a file is exported are the matrix itself and any tree descriptions contained in the data file.

Use the Export command to save data in a non-NEXUS format.

Syntax: : `export` [*options*];

Menu equivalent: **File >Export Data** ...

Available Options:

- **File** = ***export-file-name***
- **Format = Phylip|Hennig|TabText|Text|Nexus|**
- **Trees = YES|No**
- **LineBreaks = Native|Macintosh|DOSWindows|Unix**
- **CharsPerLine** = ***integer-value*****|All**
- **Interleaved = Yes|No**
- **MSTaxaHennig = BinCode|Missing**
- **UnordPhylip = Exclude|Unmodified**
- **OrdPhylip = Exclude|BinCode|Unmodified**
- ***Replace = Yes|No**

*Option is nonpersistent

Description of options

Format = Phylip|Hennig|TabText|Text|Nexus

The Format option specifies the format in which data are saved to the export file. Phylip requests data to be output to version 3.4 of Phylip package by Felsenstein (1991). Hennig requests the output format for version 1.5 of the Hennig86 program by Farris (1988). Data can also be saved as either tab delimited text (Tabtext) or simple text (Text). Nexus requests that the data be stored in the standard NEXUS Taxa and Characters blocks.

Trees = Yes|No

If either the Phylip or Hennig format is chosen Trees = Yes will save the trees currently in memory and the data set to the export file.

LineBreaks = Native|Macintosh|DOSWindows|Unix

Specifies the line termination used in the output file.

CharsPerLine = *integer-value*|All

Specifies the number of characters written per line.

Interleaved = Yes|No

If Interleave = Yes, data are stored in "blocks" , which contain information for some of the characters and all of the taxa.

MSTaxaHennig = BinCode|Missing

UnordPhylip = Exclude|Unmodified

OrdPhylip = Exclude|BinCode|Unmodified

File = *export-file-name*

Specify the export-file-name containing non-NEXUS-formatted data. (See "Input/Output files" for details regarding the file-name format).

*Replace = Yes|No

If Replace is not specified explicitly and the file already exists, you will receive a warning and will have the opportunity to cancel the command before the contents of the existing file are erased.

Note: *All other PAUP/MacClade instructions are not included.*

For example, the following is a sample data matrix with one user-tree and two taxa transferred to the outgroup:

```
#NEXUS
begin data;
  dimensions ntax=8 nchar=16;
  format missing=?;
  matrix
taxonA 000000100101?000
taxonB 0100000011011111
taxonC 111000001101?000
taxonD 111110101101?000
taxonE 1101100111011000
taxonF 111101011111?111
taxonG 101111101011?000
taxonH 101111101011?000
  ;
end;

begin trees;
 translate
  1 taxonA,
  2 taxonB,
  3 taxonC,
  4 taxonD,
  5 taxonE,
  6 taxonF,
  7 taxonG,
  8 taxonH
  ;
 utree PAUP_1 = (1,(2,(3,(((4,(7,8)),5),6))));
end;

begin PAUP;
  outgroup taxonA taxonB;
end;
```

When this is converted to Hennig86 format, it appears as:

```
  xread
  'File "test" converted for Hennig86 by PAUP'
  16 8
  taxonA
  000000100101?000
  taxonB
  0100000011011111
  taxonC
  111000001101?000
  taxonD
  111110101101?000
  taxonE
  1101100111011000
  taxonF
  111101011111?111
  taxonG
  101111101011?000
  taxonH
  101111101011?000
  ;

  tread
  '1 tree(s) from PAUP'
  (taxonA taxonB (taxonC (((taxonD (taxonG taxonH))
taxonE)taxonF)));
  procedure /;
```

Notice that the outgroup designation in the PAUP data file does not get exported-you must reset that yourself, either within Hennig86 or by editing the Hennig86 data file. The PHYLIP equivalent of this file looks like:

```
  8 16
  taxonA 000000100101?000
  taxonB 0100000011011111
  taxonC 111000001101?000
  taxonD 111110101101?000
  taxonE 1101100111011000
  taxonF 111101011111?111
  taxonG 101111101011?000
  taxonH 101111101011?000
  1
```

```
(taxonA,taxonB,(taxonC,(((taxonD,(taxonG,taxonH)),taxnE),taxonF)));
```

Excluded and ignored characters are not exported. To export all characters in the matrix, include any excluded characters and reset the ”ignore” option to ”ignore none”.

The options for importing are more varied. Besides PHYLIP, Hennig86, and tab-delimited text, you can select two formats for molecular data: GCG MSF or NBRF-PIR. If PHYLIP is chosen, you have the option of selecting discrete data, DNA sequence, restriction site data, or protein sequence. Other data formats can be imported by selecting text in the **Import File** dialog box. You can then select from standard format, DNA, RNA, or protein. Using this option gives you great flexibility in analyzing molecular data sets in a wide variety of formats.

3.8 Getting help

PAUP* is equipped with a built in help center. Users may find this especially helpful when using command line versions of the program. The ? command is a synonym for Help. ? with no arguments requests a list of the available commands.

Command-line: [*PAUP command*] ?

Menu equivalent: **Help >Paup help...**

The user may also use the HELP command to obtain help using PAUP* command-line.

Syntax: : `Help` [*Commands|command-name*]

If invoked with no arguments, Help produces a list of the available commands. If Commands (or Cmds) is specified, a one-line description of each command is output. If a command-name is specified, Help provides information of that command.

Example 1:

The following example requests a lists of available commands.

```
help;
```

Example 2:

The following example requests a list of available commands, with a one-line description of each.

```
help commands;
```

Example 3:

The following example requests help on the branch and bound (Bandb) command.

```
help bandb;
```

3.9 Logging results

Ordinarily, you will want to log the results of a PAUP* session to a disk file to have a record of the results of your analyses.

By default, output generated by PAUP* goes only to the "display buffer," a region of memory set aside exclusively for this purpose. PAUP*'s main display window is used to view this information. The Log command may be used to request direction of PAUP output to a file (e.g., for subsequent printing).

Syntax: : `Log` [*options*];

Menu equivalent: **File >Log Output to Disk** ...

Example:

The following example uses the Log command to save basic information about the trees (see the "DescribeTrees" command) in memory to the file named "myoutput.log"; after which logging is stopped. In the final line logging is resumed and the subsequent output is appended to the original log file.

Command-line

- Type: `log file=myoutput.log;`
- Type: `describetrees;`
- Type: `log stop;`
- Type: `log start append;`

Menu equivalent

- Select **File >Log Output to Disk...**
- **Type:** `myoutput.log` in **Log subsequent output to:** field and click **Start Saving**
- Select **Trees >Describe Trees...** click **Describe**
- Select **File >Log Output to Disk...** and click **Stop Saving**
- Select **File >Log Output to Disk...** click **Start Saving**. Since `myoutput.log` already exists, you will be prompted to **Replace** or **Append**. Click **Append**

Available Options:

- **File** = ***log-file-name***
- ***Replace** = **Yes|No**
- ***Append** = **Yes|No**
- ***Start** = **Yes|No**
- ***Stop** = **Yes|No**
- **FlushLog** = **Yes|No**

*Option is nonpersistent

Description of options

File = *log-file-name*

The name and location of the the log file may be specified using the File command. If the file-specification contains any of the characters equal-sign (=), semicolon (;), colon (:), or blank, it must be enclosed within single-quotes. See ”Input/Output files” on for information on how to specify platform specific path names.

Replace = Yes|No

If Replace is not specified explicitly and the file already exists, you will receive a warning and will have the opportunity to cancel the command before the contents of the existing file are erased.

Append = Yes|No

If Append is specified, subsequent output is appended to the previous contents (if any) of the file. Otherwise, subsequent output will overwrite the original contents of the file. The Append/Replace setting is retained between invocations of the Log command, unless the file is changed by a File= directive.

Start = Yes|No

If Start is specified logging is initiated to the named file. If File is not specified explicitly, a default name is assigned.

FlushLog = Yes|No

Specification of Flushlog causes the file’s buffer to be flushed after every line of output. Ordinarily, this degrades system performance and is not recommended. However, there may be situations in which immediate flushing is useful.

3.10 Issuing a time stamp

PAUP* allows the user to stamp an output or log file with the current time and date.

Use the Time command to output the current time and date.

Syntax: : `Time;`

3.10.1 *Terminating a data file*

Use the Leave command to terminate processing of an input file. Ordinarily, PAUP* continues processing until the end of an input file is reached. If, for whatever reasons, you do not want some of the commands in the input file to be processed, insert a Leave command into the file at the point where you want execution to stop. PAUP* then continues as if the end of the file were reached at that point. Leave has no effect if issued interactively.

Syntax: : `Leave;`

3.11 *Quiting the program*

The Quit command causes PAUP* to terminate. Unless you specify the option Warntsave = No, PAUP* will prompt you before quitting if there are unsaved trees in memory. Warntsave is the only option under this command.

Syntax: : `Quit` [*options*];

Menu equivalent: **File >Quit**

Available Options:

- **WarnTSave = Yes|No**

4 Managing Your Data

4.1 Introduction

PAUP* allows users to analyze a wide variety of data types (e.g. morphological, DNA, and amino acid data). The program allows the user to define weights and to characters, or sets of characters, in addition to defining ancestral states. The program also allows the user to easily exclude and include characters and taxa, as well as subsets of the data. PAUP* has the capability of predefining ”sets” and ”partitions” of characters and taxa which further facilitates the ease at which they can be manipulated. The user also has the ability to view the current stats of how data is being treated. PAUP* also allows the user to define a subset of taxa as an outgroup, which becomes important in visualizing trees (see ”Manipulating and Summarizing Trees”) as well as interpreting character evolution.

4.2 Excluding and Including Characters

Excluding characters is actually a specialized version of weighting characters-excluded characters are simply assigned a weight of zero. Excluded characters therefore do not contribute to overall tree lengths. However, they are still used in other contexts; for example, it possible to examine character-changes in excluded characters with the Describe and

ChgPlot commands. An increasingly common approach is to draw inferences about the evolution of one set of characters based on a phylogeny computed from an independent second set of characters. In PAUP*, this can be accomplished simply by excluding the first set of characters, searching for trees based on the second set, and then using the standard facilities for interpretation of character changes in the first set.

4.2.1 *Excluding characters*

Use the Exclude command to exclude one or more characters from tree-score calculations.

Syntax: `Exclude character-list [/ONLY]`

Unless /ONLY is specified, characters specified in the character-list are simply added to the set of currently excluded characters. If some characters have already been excluded but you want only those characters specified in character-list to remain excluded, specify /ONLY; any currently excluded characters not explicitly specified in the list will be re-included. PAUP* includes several predefined characters that may be used in conjunction with or without a character-list. See ”CharSet” command for more details.

Menu equivalent: **Data > Include-Exclude Characters...**

The Exclude command is used to specify a list of characters to be excluded.

Example:

Command-line

- Type: `exclude 1-25;`

Menu equivalent

- Select **Data > Include-Exclude Characters ...**
- Select character 1 through 25, click **>>Exclude >>**
- Click **OK**

This requests exclusion of the first 25 characters.

4.2.2 Restoring deleted characters

Use the Include command to re-include characters that were previously excluded.

Syntax: `Include character-list [/Only];`

Unless /Only is specified, characters specified in the character list are simply removed from the set of currently excluded characters. If you want only those characters specified in the list to be included, specify /Only; characters not explicitly specified in the list will then be excluded. PAUP* includes several predefined characters that may be used in conjunction with or without a character-list. See ″CharSet″ command for more details.

Menu equivalent: **Data > Include-Exclude Characters...**

If you want to restore previously excluded characters to the analysis, you can use the Include command.

Example:

Command-line

- Type: `include 5-15;`

Menu equivalent

- Select **Data > Include-Exclude Characters...**
- Select characters 5-15 from the **Excluded characters:** list.
- Click **<<Include <<**
- Click **OK**

restores ten characters to their original nonexcluded status.

Ordinarily, the effect of the Exclude command is cumulative; i.e., any characters already excluded by prior Exclude commands remain excluded when a new Exclude command is issued. If you want only the characters specified a single Exclude command to be excluded, you can use the Only option:

```
exclude 6 8 11/only;
```

This is equivalent to the pair of commands:

```
include all;

exclude 6 8 11;
```

4.3 Defining and Applying Character Weights

By default, PAUP* assigns equal weight to each character in the data matrix. You may, however, prefer to attach greater weight to some characters than to others. When you assign character weights, PAUP* simply computes a weighted sum of the single-character tree lengths when calculating the total length of the tree.

4.3.1 Assigning Weights

You can use the Weights (= Wts) command to specify a priori character weights.

Syntax: `weights [options]`

Menu equivalent: **Set > Character Weights**

Example:

```
weights 2:4-10 15 18-20, 3: 11 12;
```

assigns a weight of 2 to eleven characters and a weight of 3 to two characters. If a weight is not explicitly assigned to a character, the character retains the weight that was previously in effect; i.e., the effect of the Weights command is cumulative. The sequence of commands

```
wts 2:1-5;
wts 2:10;
```

results in the first five and the tenth characters having a weight of 2.

If a more than one weight is assigned to a given character, the last assignment applies. This precedence rule can be used to advantage if you want to assign a nonunit weight to most of the characters. For example, the command

```
weights 2:all, 1:2 6 11;
```

assigns weights of 2 to all characters other than 2, 6, and 11, which receive a weight of 1. Likewise, if you want to be sure that only the characters specified in a single Weights command receive nonunit weight, you could issue a command such as

```
weights 1:all, 2:5-15 31-45;
```

Any nonunit weights previously in effect are then reset to unity before a weight of 2 is assigned to some of the characters.

It is important to realize that the assumption of ″equal weights″ does not necessarily mean that each character has the same influence in discriminating among alternative tree topologies. In general, the greater the number of states observed for a character, the greater will be that character's influence (e.g., a character with five discrete states will always contribute at least four steps to the tree length, whereas a binary character may contribute as little as one step). This issue is particularly relevant if the number of states recognized is arbitrary, as for a continuous character broken into an arbitrary number of discrete states. In this case, a more ″fine-grained″ coding will lead to more character states and therefore more influence on the analysis than will a relatively ”coarse-grained” coding. You can use the Scale option to request that weights be assigned such that the minimum possible length for each character is the same for all characters in a group. The simplest use of this option is as follows:

```
weights scale;
```

In this case, weights are assigned to all characters such that the minimum possible length of each character is 1000 (the default ”base weight”). That is, binary characters are assigned a weight of 1000, three-state characters a weight of 500, and so on.

NOT TRUE: Because version 3.1 of PAUP uses integers to store tree lengths and character weights, fractional weights such as 0.5. 0.33, etc., can no longer be used in PAUP*. You can specify a base weight other than 1000 by using the BaseWt option. For example, the command

```
wts scale/basewt=100;
```

requests a base weight of 100, provides results equivalent to using decimal weights recorded to 2 decimals. A slight problem with the use of a number like 100 or 1000 as the base weight is roundoff error. For example, if your data contains a mixture of two-, three-, and four-state characters, the scaled weights resulting from a base weight of 1000 are 1000, 500, and 333, respectively. Suppose two trees are being evaluated and that exactly one character is homoplastic (i.e., requires extra steps) on each tree. Further suppose that the first tree requires three extra steps (homoplasies) in a four-state character and that the second tree requires two extra steps in a three-state character. Even though the two trees should be considered equally parsimonious, the first tree is deemed shorter, since 3 x 333 = 999 is less than 2 x 500 = 1000. In this case, using a base weight of 6 solves the problem; the scaled weights then become 6, 3, and 2, with 3 x 2 = 2 x 3.

You can mix scaled and unscaled weights in the same command. For example, you might want to scale the weights for just a few characters, but use ”equal weighting” for the remaining characters. In this case, the syntactically valid command

```
wts scale:3 9-11 14;
```

would probably not produce the desired results, since the specified characters would receive weights of 1000, 500, 333, etc., but the remaining characters would retain their initial weights (probably 1). Instead, you could use a command such as

```
wts 100:all, scale/basewt=100:3 9-11 14;
```

4.4 Assumption Sets

Assumption sets allow rapid assignment of character type and weight assumptions. Each assumption set specifies a character type ("Type Sets"), weight ("Weight Sets") or exclusion status ("Exclusion Sets") for each character. Any number of type, weight, or exclusion sets may be defined. You can then change the assumptions assigned to all characters in the data set simply by invoking a new assumption set.

Assumption sets are normally defined in the Assumptions block, however you may also define them from within a Paup block or via the command line.

4.4.1 *Invoking assumption sets*

Use the Assume command to invoke a type set, weight set, or exclusion set, or to select an ancestor. The options under this command are described in the "Commands used in the Assumptions Block".

Syntax: `Assume [Options];`

Menu equivalent: **Data > Choose Assumptions Sets...**

Available Options:

- **TypeSet** = *typeset-name*
- **WtSet** = *wtset-name*
- **ExSet** = *exset-name*

- **AncStates** = ***ancstates-name***

The TypeSet, WtSet, and ExSet commands merely define assumption sets; they do not cause the specified types, weights, or exclusion status to go into effect. An assumption set must be invoked before it actually takes effect. There are three ways to invoke an assumption set:

- By preceding the type-set, weight-set, or exclusion-set name with an asterisk. For example, if the following commands were issued in sequence:

```
typeset one = ord:5-8;
typeset *two = dollo:all;
typeset three = irrev:1-5 dollo:6-.;
```

three type sets would be defined, and all characters would currently be assigned type Dollo.

- By using the Assume command. This command allows you to invoke any combination of type sets, weight sets, exclusion sets, and ancestors. For example:

```
assume typeset=one wtset=mywts
exset=noncoding;
```

sets the current character types, weights, and exclusion status to the settings defined in the type set ”one,” the weight set ”mywts,” and the exclusion set ”noncoding,” all of which must previously have been defined in TypeSet, WtSet, and ExSet commands, respectively.

- By using popup menus in the dialog boxes associated with the **Data > Set Character Types**

  , **Data > Set Character Weights**

  , and **Data > Include-Exclude Characters**

  menu commands.

4.4.2 *Type sets (”TYPESETs”)*

Type sets assign a character type to each character. For example, if you wanted the first 10 characters to be of type Unord, the second 10 to be of type Dollo, and the remainder to be of type Ord, you could define a corresponding type set as follows:

```
typeset mytypes = unord:1-10, dollo:11-20, ord:21-.;
```

If you do not explicitly assign a type to one or more characters, the current default character type (specified via DefType in the Options command of the Assumptions block) will apply. Note that the default DefType is Unord, so ordinarily you will only need to assign types to those characters to which you want to assign a type other than Unord.

Be careful to use the correct punctuation. If, for instance, you omitted the first comma in the above example, the character-type dollo would be interpreted as a character identifier, and would generate an error message.

Remember that you can use character sets (see above) to refer to groups of characters in the TypeSet command. Simply specify a character-set name anywhere a character name or number could be used.

4.4.3 Weight sets (WTSETs)

Weight sets assign a weight to each character. For example, you might define a weight set as follows:

```
wtset mywts = 2: 2 4 8-12, 3: 6 14-20;
```

Any character for which a weight is not explicitly assigned receives a weight of one. Be careful to use the correct punctuation. If, for instance, you omitted the first comma in the above example, the next "3" would be taken as a character number and assigned weight 2. However, the immediately following colon would generate a syntax error message.

An alternate format, "vector", allows you to specify the weight for each character sequentially rather than by character-lists. For example, the following command is equivalent to the one shown above:

```
  wtset mywts2 vector = 1 2 1 2 1 3 1 2 2 2 2 2 1 3 3 3 3
3 3 3;
```

Remember that you can use character sets (see above) to refer to groups of characters in the WtSets command. Simply specify a character-set name anywhere a character name or number could be used.

4.4.4 Exclusion sets (EXSETs)

Exclusion sets allow you to specify a set of characters that are to be ”excluded” from the analysis (see ”Excluding Characters” under ”Character Weighting” above).

Example:

```
exset dontwant = 4 11-20 31-34;
```

specifies a list of 15 characters that can be excluded by invoking the exclusion set ”dontwant.”

Remember that you can use character sets (see above) to refer to groups of characters in the ExSet command. Simply specify a character-set name anywhere a character name or number could be used.

4.5 Deleting and Restoring Taxa

4.5.1 Deleting taxa

Use the Delete command to delete a taxon from subsequent analyses.

Syntax: `Delete taxon-list [/[ONLY][options]];`

Unless ONLY is specified, taxa specified in the taxon-list are simply added to the set of currently deleted taxa. If taxa have already been deleted and you want only those taxa specified in taxon-list to remain deleted, specify ONLY; any currently deleted taxa not explicitly specified in the list will be restored.

Menu equivalent: **Data > Delete-Restore Taxa...**

Available Options:

- * **Prune = Yes| No**
- * **ClearTrees = Yes| No**
- **Condense = Yes| No**

*Option is nonpersistent

Description of options

The following three options pertain only if there are trees in memory that will become invalidated by the deletion of taxa. If you do not specify Prune or ClearTrees, the program will prompt for your desired action. Prune and ClearTrees are mainly useful for batch file processing, where you do not want the program to stop and wait for a response from the user before continuing.

Prune = Yes|No

If Prune = Yes is specified, newly deleted taxa will be removed ("pruned") from the trees currently in memory, which otherwise remain unmodified.

ClearTrees = Yes|No

If ClearTrees is specified, any trees currently in memory are simply deleted.

Condense = Yes|No

If Condense is specified, any duplicate trees that result from the removal of taxa (Prune option) are deleted.

4.5.2 Restoring deleted taxa

Use the Undelete (=Restore) command to restore previously deleted taxa (or delete taxa not in list).

Syntax: `Undelete taxon-list [/[Only] [ClearTrees] [options]];`

Unless /Only is specified, taxa specified in the taxon list are simply removed from the set of currently deleted taxa. Specify /Only to delete, in addition, any taxon not explicitly specified in the list.

Menu equivalent: **Data > Delete-Restore Taxa...**

Available Options:

- ***Prune = Yes|No**
- ***ClearTrees = Yes|No**

Description of options

The following two options are ignored unless the "/Only" option is also requested, in which case the effect of the command may be a deletion of currently nondeleted taxa.

Prune = Yes|No

If Prune = Yes is specified, newly deleted taxa will be removed ("pruned") from the trees currently in memory, which otherwise remain unmodified.

ClearTrees = Yes|No

If CLEARTREES is specified, any trees currently in memory are simply deleted.

4.6 Defining Sets

”Sets” in PAUP* provide a way to refer to collections of objects with single names. The use of sets can greatly reduce the amount of redundant typing needed to issue commands, and can help to avoid mistakes when typing commands into the command line.

Ordinarily, you will define sets within the Assumptions block, although you may also define them from within a Paup block or via the command line. Once a set has been defined, you can use it in any subsequent command that recognizes sets of that type.

4.6.1 Character Sets (”CHARSETs”)

The use of character sets allows you to refer to a group of characters by a single name. The only restriction is that the name of the character set cannot be the same as the label assigned to an individual character, for obvious reasons. After defining a character-set using a CharSet command, you can use the character-set name in any place that you would ordinarily use a character name or number.

The CharSet command specifies and names a set of characters; this name can then be used in subsequent CharSet definitions or wherever a *character-list* is required. The `Vector` format consists of 0’s and 1’s: a 1 indicates that the character is to be included in the CharSet; whitespace is not necessary between 0’s and 1’s. The name of a CharSet cannot be equivalent to a character name or character number.

Use the CharSet command to define a ”character set.” Character sets are simply groups of characters that can be referred to by a single name in other commands.

4.6.2 Defining character sets

Syntax: `CharSet character-set-name=character-list;`

Table 4.1 Predefined Character Sets and Their Corresponding Data Types

Date Type	Character Set Name	Definition
All data types	`constant`	Invariant characters
	`gapped`	Characters with a gap for at least one taxon
	`missambig`	Characters with a gap or ambiguous character for at least one taxon
	`allmissing`	Character with gap for all taxa
	`uniform`	Characters that are constant for all taxa
	`remainder`	Character not previously referenced in the command
DNA, RNA, and Nucleotide	`pos1`	Characters defined by current CodonPosSet as first positions
	`pos2`	Characters defined by current CodonPosSet as second positions
	`pos3`	Characters defined by current CodonPosSet as third positions
	`noncoding`	Characters defined by current CodonPosSet as non-protein-coding

The *character-set-name* must not be identical to any of the original character names. For example, you could define two character sets as follows:

```
  charset larval=1-10 26-30;

  charset adult=11-25 31-50;
```

and then use the character-set names in subsequent commands:

```
  exclude larval; [excludes larval characters]

  weights 1:larval, 2:adult; [assigns double-weight to
adult characters]
```

The preceding two commands are exactly equivalent to the commands:

```
exclude 1-10 26-30;
```

```
weights 1:1-10 26-30, adult:11-25 31-50;
```

Note that you may freely mix character names and/or numbers with character-set names. For example, the command

```
weights 2:larval 12 14 45-49;
```

assigns a weight of 2 to all of the larval and some of the adult characters.

Character sets are especially useful when analyzing molecular sequence data. For example, you might subdivide the regions of a protein-coding gene as follows:

```
charset coding = 25-99 138-234 251-298;
charset introns = 100-137 235-250;
charset flanking = 1-24 251-276;
charset noncoding = introns flanking;
charset 1stPos = 25-97\3 138-232\3 251-296\3;
charset 2ndPos = 26-98\3 139-233\3 252-297\3;
charset 3rdPos = 27-99\3 140-234\3 253-298\3;
```

You could then easily refer to these regions as in the following examples (which are not necessarily intended to be biologically meaningful):

```
exclude flanking ; [excludes flanking regions]
```

```
  wts 4:1stPos 2ndPos, 2:3rdPos, 1:noncoding; [assigns
double-weight to 1st and 2nd positions, half-weight to
noncoding positions]
```

```
  ctype tv3:coding;[assign a user-define character type to
characters in the coding regions]
```

Character-sets are also available in any dialog box that contains a character-selection list. Popup menus allow you to instantly select all of the characters in a set.

4.6.3 Taxon Sets (”TAXSETs”)

The use of taxon sets allows you to refer to a group of taxa by a single name. After defining a character-set using a TaxSet command, you can use the taxon-set name in any place that you would ordinarily use a taxon name or number. For example, you could define two taxon sets as follows:

```
  taxset myGenus=1-15 26-40;
  taxset otherSpp=16-25;
```

and then use the taxon-set names in subsequent commands:

```
  delete otherspp; [deletes taxa other than those in
'mygenus']
  outgroup otherSpp; [assigns taxa 16-25 to the outgroup]
  constraints ingrp = ((myGenus)); [define an ”ingroup
monophyly” constraint]
```

You cannot declare a TaxSet name that is the same as any taxon name in the data file.

4.6.4 Codon position sets (CODONPOSSETs)

The Codon block contains information about the genetic code, the regions of DNA and RNA sequences that are protein coding, and the location of triplets coding for amino acids in nucleotide sequences. Numerous other commands used by other NEXUS-compliant programs may be included in the Codons block. Currently only the CodonPosSet command is implemented in PAUP*; all other Codons block commands will be ignored.

```
    Begin Codons;}
      CodonPosSet [*]name [({Standard}|Vector})] =
         N:character-set,
         1:character-set,
         2:character-set,
         3:character-set;]
    End;
```

The CodonPosSet command stores information about protein coding regions and the codon positions of nucleotide bases in protein-coding regions.

Those characters designated as 1, 2, or 3 are coding bases which are specified as being of positions 1, 2, and 3, respectively. Those characters designated as N are considered non-protein-coding. Those characters designated as ? are of unknown nature. Any unspecified bases are considered of unknown nature (equivalent to ?). If no CodonPosSet statement is present, all bases are presumed of unknown nature.

For example, the following command

```
codonposset * coding=
  N:1-10,
  1:11-\3,
  2:12-\3,
  3:13-\3;
```

designates bases 1-10 as noncoding, and positions of the remaining bases in the order 123123123....

4.6.5 *Rate sets (RATESETs)*

Often the user will want to define a specific rate for each subset of the data. Use the RateSet command to define a "Rate Set".

Syntax: `RateSet rate-set-name = rateset-definition ;`

Example 1:

In the following example, the RateSet command is used to define three gene fragments according to their position in the data matrix.

```
rateset genes=1:1-300, 2:301-600, 3:601-700;
```

Example 2:

In the following example, the RateSet command is used to defined codon positions base on previously defined character sets (see the "CharSet" command).

```
  charset 1stpos = 2-457\3 660-896\3;
  charset 2ndpos = 3-457\3 661-896\3;
  charset 3rdpos = 4-457\3 662-\3;
  rateset codons=1:1stpos, 2:2ndpos, 3:3rdpos;
```

Example 3:

Rate sets are only applicable during maximum likelihood analyses. The following example defines a specific rate for each subset of the data as defined by the RateSet command.

```
  rateset genes=1:1-300, 2:301-600, 3:601-700;

  lset rates=sitespec siterates=rateset:genes;
```

(see the "LSet" command under "Setting a model for maximum likelihood calculations" for more detail on using "RateSet".)

4.6.6 Viewing rate sets

Use the ShowRateSet command to show one or all rate-set definitions.

Syntax: `ShowRateSets [rate-set-name];`

Menu equivalent: **Data > Show Other > Rate Sets**

Example:

In the following example a RateSet (see above) has been defined in a Paup block as:

```
  rateset genes=1:gene1, 2:gene2, 3:gene3, 4:gene4,
5:gene5, 6:gene6;
```

Command-line

- Type: `showratesets genes`

Menu equivalent

- Select **Data > Show Other > Rate Sets**
- Select **genes**

Issuing this command will produce the following table to be output:

```
                  111111111122222222223333333333444444444455555555556
Rate     123456789012345678901234567890123456789012345678901234567890
---------------------------------------------------------------------
1.000000 ************************************************************
2.000000 ............................................................
3.000000 ............................................................
4.000000 ............................................................
5.000000 ............................................................
6.000000 ............................................................


...


Rate-set ”genes” (continued):

         55555555
         44444444
         22333333
Rate     89012345
-----------------
1.000000 ........
2.000000 ........
3.000000 ........
4.000000 ........
5.000000 ........
6.000000 ********
```

For each character in the matrix there will be a column the number of rows corresponds to the number of predefines rate partitions (in the previous example, 6). An asterisk (*) denotes which rate partition each character has been assigned to. Depending on the number of characters in the data set, this command can produce a lot of output. There are no additional options for this command.

4.6.7 *Character status*

Use the CStatus command to request a listing of character-status information for all characters.

Syntax: `CStatus [options];`

Menu equivalent: **Data > Show Character Status**

Example:

The following example will provide a brief summary of the character status of the characters loaded in to memory. Once again, depending on the number of characters a data set, issuing the command CStatus Full=Yes may result in a considerable amount of information being printed to the screen.

Command-line

- Type: `cstatus full=no`

Menu equivalent

- Select **Data > Show Character Status**
- Choose **Brief summary**

Available Options:

- **Full = Yes| No**
- **Excluded = Show| Hide**

Description of options

Full = Yes|No

For each character, the following information is output if Full = Yes:

1. The number and name (if any) of the character.
2. The character's current type.
3. If the character is constant (invariant), excluded, or uninformative.

4. The character's current weight. The weight value is put into parentheses if the character is excluded. A list of the states observed for the character.

Excluded = Show|Hide

If Exclude = Hide then only the status of the currently included characters will be displayed.

4.7 Defining Partitions

The partition divides the objects into several (mutually exclusive) subsets. They follow the same format, and will be described together.

There are several formatting options. The Vector format consists of a list of partition names. By default, the name of each subset is a NEXUS word (this is the Tokens option). The Notokens option is only available in the Vector format; this allows one to use single symbols for the subset names. Each value in a definition in Vector format must be separated by whitespace if the names are tokens, but not if they are NoTokens.

4.7.1 Character partitions

Use the `CharPartition` to define a partition of the characters.

Syntax: `CharPartition partition-name=partition-definition;`

The CharPartition command is ordinarily issued from within the Sets block. However, you may also issue it from the command line or from within a Paup block. The following example of the CharPartition command creates a character partition named gfunc which defines coding and noncoding regions of the sequences.

Example 1:

```
  charpartition gfunc = 1:2-457 660-896, 2:1 458-659
897-898;
```

Example 2:

The following example of the CharPartition command is equivalent to the Example 1 except that two predefined character sets are used to define the characters included in each partition. See the ”CharSet” command for a complete list of predefined character sets.

```
charset coding = 2-457 660-896;
charset noncoding = 1 458-659 897-898;
charpartition gfunc = 1:coding, 2:noncoding;
```

Once again, depending on the number of characters in your data matrix, issuing this command can produce a lot of information to be output to the screen. There are no additional options for this command.

4.7.2 *Taxon partitions*

You may also define a partition of the taxa.

Syntax: TaxPartition *partition-name* = *partition-definition;*

The TaxPartition command is ordinarily issued from within the Sets block. However, you may also issue it from the command line or from within a Paup block. TaxPartition are need for ”convexity” constraints.

Example 1:

The following two examples are equivalent:

```
taxpartition populations=1:1-3, 2:4-6, 3:7 8;

taxpartition populations (vector notokens) = 11122233;
```

Example 2:

The following two examples are equivalent:

```
  taxpartition mypartition=Chiricahua:1-3, Huachuca:4-6,
Galiuro:7-8;
```

```
  taxpartition mypartition (vector) = Chiricahua
Chiricahua Chiricahua
  Huachuca Huachuca Huachuca Galiuro Galiuro;
```

4.7.3 Displaying character partitions

Use the ShowCharPartitions command to show one or all character-partition definitions.

Syntax: `ShowCharParts [partition-name];`

If no partition-name is specified, all defined partitions are output.

Menu equivalent: **Data > Show Other > CharPartitions**

4.7.4 Displaying taxon partitions

Use the ShowTaxPartitions command to show one or all taxon-partition definitions.

Syntax: `ShowTaxParts [partition-name];`

Menu equivalent: **Data > Show Other > TaxPartitions**

4.8 Multistate Taxa

Ordinarily, you will assign a unique (singleton) character-state to each taxon for each character. However, two situations may necessitate the assignment of multiple character states to a taxon:

- You may be uncertain about the state that a particular taxon possesses, but some of the potential states can be excluded as possibilities. This condition is called ″partial uncertainty.″
- A terminal taxon may be an assemblage of lower-level taxa which vary in the character-state possessed. This condition is loosely referred to as ″polymorphism.″

When multiple states are interpreted as ″uncertainty,″ PAUP* will choose a state from the set of available states that allows minimization of the tree length. When multiple states are treated as ″polymorphism″, PAUP* assumes that the ″terminal taxon″ is actually a heterogeneous group. In this case, all but one of the states in the polymorphic terminal taxon must be derived from a monomorphic ancestral taxon in the most parsimonious way possible. In PAUP*, you can choose either of the above interpretations of multistate taxa. However, it is not possible to mix the two interpretations at the same time (i.e., some multistate taxa interpreted as uncertainty, others as polymorphism). This current limitation will hopefully be eliminated in a future version.

IMPORTANT: ″Polymorphism″ refers only to variability within a ″terminal taxon.″ Multistate taxa do not provide a mechanism for dealing with characters that are polymorphic in a population-genetic sense. In particular, there is no provision for polymorphism in hypothetical ancestral taxa (internal nodes). If a taxon is coded as having multiple states, it is assumed that this taxon represents a monophyletic collection of subtaxa, each of which are themselves monomorphic. The program then assumes that the ancestor of this monophyletic group possessed one of the observed states, from which the other states were subsequently derived. Thus, for example, if two sister taxa were both coded as being polymorphic for states 0 and 1, the ancestor of this pair of taxa would be assigned either 0 or 1 (in order to minimize the overall number of changes on the tree), and either the 0′s or the 1′s would be interpreted as parallelisms. While it might seem reasonable to assign both states to the ancestor with the polymorphism being retained in the two descendant taxa, this would require additional assumptions about the relative probabilities of retention of polymorphism vs. character transformation that are beyond PAUP*′s scope.

To assign multiple states to a taxon, enclose all of the desired states within curly braces or parentheses, as in the following example:

```
matrix
 tax1    1 1    0 0
```

```
    tax2    1 {12} 1 0
    tax3    0 2    1 (01)
    tax4    0 0    1 1
    ;
```

PAUP* does not distinguish between the curly braces and parentheses. However, in MacClade, multiple states enclosed in curly braces imply the ″uncertainty″ interpretation, whereas those enclosed in parentheses imply polymorphism. Thus {12} in the above example would imply ″state 1 or state 2,″ whereas the (01) would imply ″both states 0 and 1.″ For compatibility with MacClade (and in anticipation of future versions of PAUP*) you may want to adopt this convention now.

In some situations, it may be helpful to use ″equate macros″ to enter multistate taxa into the data matrix. The replacement of character-state specifications of the form `{abc}` or `(abc)` by a single character allows easier alignment of columns of the data matrix. For example, the matrix above could be equivalently defined as follows:

```
format equate=”a=(12) b=(01)”;
matrix
 tax1 1100
 tax2 1a10
 tax3 021b
 tax4 0011
 ;
```

Note to MacClade Users: *The above-mentioned distinction between PAUP* and MacClade is subtle. If you created your PAUP* input file using MacClade's spreadsheet editor and chose either the ″and″ (by default, ″&″) or ″or″ (by default, ″/″) separator to indicate the two possible interpretations of multistate taxa, then MacClade would export 0 & 1 as (01), and 0/1 as {01}. Likewise, MacClade respects the distinction between (01) and {01} on import. To choose between these differing interpretations of multistate taxa in PAUP*, you must use one of the methods described above.*

In practice the difference between designating multistate taxa ″polymorphic″ or ″uncertain″ will be manifested in some constant difference in tree length. Trees calculated using the ″polymorphic″ option will be longer because they force extra change within the multistate taxon (see above). When the ″uncertain″ option is used, PAUP* chooses the state that leads to a minimal tree, and does not invoke any further change within the multistate taxon.

Complications occur when a taxon is multistate for a Dollo or irreversible character. In those cases, PAUP* will select the state which minimizes overall tree length and satisfies the demands of the character type. With Dollo characters, PAUP* will allow a single derivation and multiple reversals, if necessary. With an irreversible character, PAUP* will allow multiple derivations, but no reversals. In the case of uncertainty, PAUP* chooses the state from the multistate taxon which satisfies the character type and minimizes tree length. So PAUP* ensures that the character type is maintained over the tree, and can choose the best character state from the ″uncertain″ multistate terminal. It is even possible that a character for which one or more taxa is multistate may not require any extra steps if that character is treated as ″uncertain.″ However, when the multistate taxon is ″polymorphic″ for a Dollo or irreversible character, the best fit will be one or more derivations of each state internally plus whatever is required within the terminal, depending on the type of character (remember that PAUP* must explain all of the states in a ″polymorphic″ multistate taxon). In the case of a Dollo character, unique derivation and reversal but not multiple derivation can be tolerated; in the case of an irreversible character, independent derivation but not reversal are allowed. Because PAUP* is required to account for all the states in a ″polymorphic″ taxon and to adhere to the demands of the character type over the entire tree, the states PAUP* assigns internally can be very different from those that would be assigned under a condition of ″uncertainty.″ This of course can lead to different tree topologies, so the difference between treating multistate taxa as ″polymorphic″ versus ″uncertain″is certainly non-trivial. Keep this in mind when you include multistate taxa in your analysis.

4.9 Defining and Using Outgroups

Use the Outgroup command to assign one or more taxa to the outgroup.

Syntax: `Outgroup taxon-list [/Only];`

Unless /Only is specified, taxa specified in the taxon-list are simply added to the current outgroup. If taxa have already been assigned to the outgroup and you want only those taxa specified in taxon-list to remain in the outgroup, specify /Only; any taxon that is not explicitly specified in the list will be transferred to the ingroup.

Menu equivalent: **Data > Define Outgroup...**

Unless you are using directed characters or you choose to include an ancestral taxon, the trees found by PAUP* searches are unrooted (see ”Outgroups, Ancestors, and Roots”). The most commonly used method for rooting these trees is to include an assumed outgroup in the analysis. You can use the Outgroup command to define an outgroup. For example, the command

```
outgroup gar shark;
```

specifies that the taxa ”gar” and ”shark” are to be considered outgroup taxa relative to the remaining ingroup.

You can also assign outgroup taxa by number, e.g.:

```
outgroup 1 18-21;
```

The taxon numbers correspond to the row numbers of the taxa in the data matrix.

If you want to move a taxon previously assigned to the outgroup back to the ingroup, you can use the Ingroup command. For example:

```
ingroup gar;
```

returns the taxon ”gar” to the ingroup.

Ordinarily, the effect of the Outgroup command is cumulative; i.e., any taxa already assigned to the outgroup by prior Outgroup commands remain as outgroups when a new Outgroup command is issued. If you want only the taxa specified in a single Outgroup command to be assigned to the outgroup, you can use the ONLY option:

```
outgroup 18/only;
```

This is equivalent to the pair of commands:

```
ingroup all;
outgroup 18;
```

4.9.1 *Restore taxa to the ingroup*

Use the Ingroup command to return one or more taxa to the ingroup.

Syntax: `InGroup taxon-list [/Only];`

Unless /Only is specified, taxa specified in the taxon-list are simply removed from the current outgroup. If you want only those taxa specified in taxon-list to be included in the ingroup, specify /Only; any taxon that is not explicitly specified in the list will be transferred to the outgroup.

Menu equivalent: **Data > Define Outgroup....**

4.10 *Examining Current Status*

PAUP* provides several commands for displaying the current status of data, trees, and ancestors.

4.10.1 *Displaying the data matrix*

To display the current data matrix, use the ShowMatrix command.

Syntax: `ShowMatrix`

Menu equivalent: **Data > Show Data Matrix**

By default, ignored/zapped characters will not be displayed. They can be displayed if Set ShowIgnore is used, or that option is selected in the **Options >Character Matrix Format...** pull-down menu dialog box. Excluded characters are always displayed. If Set CmStatus is used, uninformative and constant characters are flagged. For example, the following matrix has both ignored and deleted characters:

```
Input data matrix

                     1111111
Taxon       1234567890123456
----------------------------
 Constant:             **
 Uninf.:          *  **
 Excluded:     **
----------------------------
taxonA      000000100101?000
taxonB      0100000011011111
taxonC      111000001101?000
taxonD      111110101101?000
taxonE      1101100111011000
taxonF      111101011111?111
taxonG      101111101011?000
taxonH      101111101011?000
```

The width of each column can be controlled with the Set CmdColWid command . The default is one column (CmdColWid = 1). If ”equate” macros are used in the data file (e.g. a=12) and character-state reconstructions are requested, the ”equate” symbol will be assigned to internal nodes if Set CmShowEq is chosen. ShowMatrix will always display the ”equate” symbol. If NoCmShowEq is selected, the possible character states will be displayed instead of the ”equate” symbol.

4.10.2 *Displaying character status*

To display the status of characters, use the **Data >Show Character Status** menu command. Excluded, zapped, constant, and ignored characters are flagged. One column has a ”Y” or ”N” depending on whether the character is informative. Output also includes current character type (ordered etc.)., character weight, and observed states.

```
Current status of all characters:

Character        Type      Inform?  Status     Weight    States
```

```
1                Irrev.Up     Y                      1    01
2                Unord        Y                      1    012
3                Unord        Y                      1    01
4                Unord        Y                      1    01
5                Ord          Y                      1    012
6                Ord          Y                      1    012
7                Unord        Y                      1    01
8                Unord        Y                      1    01
9                Unord        N     Ignored          1    01
10               mytype       Y     Excluded        (1)   01
11               Unord        Y     Excluded        (1)   01
12               -            N     Const/Ign        1    1
13               -            N     Const/Ign        1    1
14               Unord        Y                      1    01
15               Unord        Y                      1    012

There are 1 ”zapped” characters
```

4.10.3 *Displaying taxa status (TSTATUS)*

Use the TStatus command to obtain information on which taxa, if any, are deleted and which taxa have been assigned to the outgroup.

Syntax: `TStatus [options];`

Menu equivalent: **Data > Show Taxon Status...**

Example:

Issuing this command will result in something similar to the following output:

```
3 unrooted trees in memory
No tree filter in effect
Source of trees: Heuristic search
```

If trees in a file are added to trees in memory, PAUP* will identify ”treefile” as the source for all trees in memory.

Available Options:

- **Full = No|Yes**
- **Deleted = Show|Hide**

Description of options

Full = No|Yes

By default (Full = No) PAUP* briefly summarizes the number of taxa, in the data matrix, deleted, and assigned to the outgroup. If Full = Yes one line per taxon is given.

Deleted = Show|Hide

Unless Delete = Hide, TStatus will display information for all of the taxa included in the Data or Taxa block.

4.11 Missing Data

Ordinarily, you will assign a state for every character in every taxon. However, in two situations you may need to score a character state as ”missing”:

- the data for a particular character are simply unavailable in a taxon, but you want to include the taxon in the analysis anyway, or
- the character ”does not apply” (e.g., the character represents different modifications of a structure that is absent entirely in some taxa).

In either case, a possible solution is to assign to the taxon the character state that *would* be most parsimonious *given its placement on the tree* (but see Maddison (1993)). Effectively, then, only those characters that have non-missing values will affect the location of any taxon on the tree. The example below illustrates this concept. Remember that for the purpose of computing tree lengths, each character can be treated independently; in this example, one character is considered in isolation from all of the others. For tree **A**, PAUP* would assign state ”1” to the taxon with missing data (= ***B***). This allows a reconstruction requiring only one step over the full tree. For tree **B**, however, assignment of state ”0” is required in order to minimize the length required by this character, again one step. Thus, both trees are equally parsimonious with respect to this character; if one tree is in fact ”better” than the other, evidence for this conclusion will have to come from other characters.

```
A   B           B  D  E   F
1  ?->1        ?->0 0 0   0
 \ /  C          \ /   \ /
  1   1           0     0
   \ /  D          \   /  A   C
    1   0           \ /   1   1
     \ /  E          0     \ /
      0   0           \     1
       \ /  F          \   /
        0   0           \ /
         \ /             0
          0               \
           \               \
            0               0
            G               G
       A               B
```

Demonstration of PAUP's handling of missing data. Reconstruction of one character on two different trees (see text).*

Note that in some cases, the assignment of the ”would-be” most parsimonious state is ambiguous. However, the length required is the same regardless of how the ambiguity is resolved, so that for the purposes of calculating the length of a particular tree or comparing the lengths of alternative trees, ambiguity does not present a problem. In the example below, either state ”0” or ”1” could be assigned to taxon B (reconstructions A and B, respectively). In either case, the required length of two steps is greater than that required by the trees above. Thus, although this character is not informative for placing taxon B, it is still useful for discriminating among tree topologies.

```
D   E        D   E
0   0        0   0
 \ /  F       \ /  F
  0   0        0   0
   \ /  A       \ / A
    0   1        0   1
     \ /  C       \ /  C
```

```
      0   1        1   1
       \ /  B       \ /  B
        0  ??0       1  ??1
         \ /          \ /
          0            1
           \            \
            0            0
            G            G
A                 B
```

Two equally parsimonious reconstructions on the same tree, demonstrating the handling of ambiguity in the presence of missing data. Although the two reconstructions are different, the length of the reconstructions is two steps in each case.

When reconstructing characters, PAUP* always assigns a nonmissing state to all internal nodes. Occasionally, users have complained about this behavior. For example, some users have argued that for the reconstruction shown below, state '?' should have been assigned to the nodes at the base of the {D,E} and {C,D,E} clades rather than state 1. This choice would imply, however, that state 0 could be assigned to these nodes in a most-parsimonious reconstruction. But if state 0 were assigned to either of these nodes, the reconstruction would require at least two steps rather than the one step implied by the indicated reconstruction. Thus, state 1 is the only permissible state for these nodes.

```
A   B  C  D   E
1   1  ?  ?   ?
 \ /   \   \ /
  1     \   1
   \     \ /
    \     1
     \   /
      \ /  F
       1   1
        \ /  G
         1   0
          \ /
           0
            \
             0
             H
```

Similarly, in the example below, we must assign either state 1 or state 2 to all internal nodes, even though non-missing states are defined only for taxa A and B.

```
A   B   C   D
1   2   ?   ?
 \ /   /   /
  \   /   /
   \ /   /
    1   /
     \ /
      1
       \
        ?
```

Sometimes, internal-node state assignments are truly ambiguous due to the presence of missing data. For example, either of the two reconstructions below require the minimum length of one step:

```
A   B   C   D         A   B   C   D
1   1   ?   0         1   1   ?   0
 \ /   /   /           \ /   /   /
  1   /   /             1   /   /
   \ /   /               \ /   /
    0   /                 1   /
     \ /                   \ /
      0                     0
       \                     \
        0                     0
```

A character-state reconstruction that illustrating the inappropriateness of assigning missing values to internal nodes.

PAUP* always chooses one of the potential reconstructions according to the character types and optimization options in effect. However, it also provides capabilities for examining all possible state assignments to internal nodes so that you can determine whether or not a reconstruction contains ambiguities.

See "Character-State Optimization" for information regarding the resolution of ambiguity in character-state reconstructions.

4.12 Nucleotide-pair frequencies

Use the PairDiff command to show nucleotide-pair frequencies.

Syntax: `PairDiff [taxon-list] [/options];`

Menu equivalent: **Data > Pairwise Base Differences...**

Available Options:

- **ShortFmt = Yes|No**
- **LongFmt = Yes|No**

Description of options

ShortFmt = Yes|No

If ShortFmt is specified, then nucleotide-pair frequencies, proportion of sites differing, and apparent transition/transversion ratio is given on a single line for each sequence pair.

LongFmt = Yes|No

If LongFmt is specified, then a 4 X 4 matrix of nucleotide-pair frequencies, proportion of sites differing, and apparent transition/transversion ratio is given for each sequence pair both before and after adjustments are made for gaps/ambiguities.

4.13 Parsimony Character Types

PAUP* implements different parsimony variants through the declaration of a **character type** for each character included in the data matrix. Available types are **ordered** (Wagner), **unordered** (Fitch), **Dollo**, **irreversible** (Camin-Sokal), and **user-defined**. Any combination of character types can be assigned to the characters in the data matrix. The character types (and weights) that are specified constitute the a priori assumptions that are in effect for a particular analysis.

Character types may be classified as either undirected or directed. An undirected character is one in which for every pair of states a and b, the "cost" in tree length is the same for the transformations $A \to B$ and $B \to A$. All of the standard character types are undirected except for irreversible. Stepmatrix characters (defined below) are undirected if and only if the stepmatrix is symmetric, i.e., $d_{ij} = d_{ji}$ for all pairs of character states *i,j* (see below). The "directedness" of the characters determines to whether PAUP* stores trees found by searches as rooted or unrooted trees. When directed characters are in effect, trees must be treated as rooted trees, because the position of the root may affect the length of the tree. On the other hand, if all characters are undirected, PAUP* ordinarily considers the trees to be unrooted, since the length of the trees is independent of the position of the root. Unrooted trees need not be explicitly rooted before requesting subsequent output. PAUP* roots trees automatically using the currently defined outgroup (or via Lundberg rooting) whenever necessary.

In addition, characters are either **polarized** or **unpolarized**. Polarized characters are those for which a state is designated that is assumed to be ancestral to all other states. If the ancestral state is not specified or is designated as "missing," the character is said to be unpolarized.[*]

A character type may formally be described as a **weighted directed graph** (**weighted digraph**). The vertices of the graph correspond to character states, and the edges of the graph are arrows corresponding to permissible character-state changes. The weight of each edge is the "cost" associated with the transformation from one character state to another.

4.13.1 Ordered (Wagner) Characters

"Ordered" characters are those typically associated with "Wagner parsimony." The character states are ordered according to their position in the "Symbols list."

See "The Data Block" for information on the Symbols list.

[*]Unfortunately, this terminology is unstandardized. Meacham (1984) and others use "directed" vs. "undirected" in the same sense that I use "polarized" vs. "unpolarized". My usage of "directed" is more consistent with graph theoretic concepts (see next paragraph)..

The list of symbols represents a linear transformation series. For example, if `symbols="ABCDE"` were specified in the Format command of the Data block, PAUP* would treat ordered characters under the assumption that to get from state A to state E, the character must proceed progressively through states B, C, and D, as indicated in the character-state graph below:

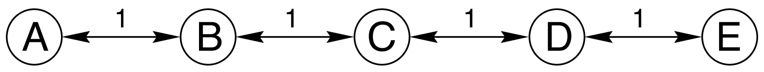


Figure 4.1 *Character-state graph for a 5-state ordered character.*

No numerical or alphabetical order is assumed. For example, if you specify `symbols="021"`, state 2 is assumed to be intermediate between states 0 and 1. Similarly, if you specify `symbols="ABQC"`, state B lies between A and Q, and state Q lies between B and C. No polarity is implied by the symbols list, however. All of the following transformations are consistent with the symbols list `"012"`; there is no requirement that '0' be the ancestral state:

state 0 ancestral: 0 → 1 → 2
state 1 ancestral: 2 ← 1 → 0
state 2 ancestral: 2 → 1 → 0

4.13.2 Unordered Characters

Unordered characters are defined such that any state is capable of transforming directly to any other state, with equal cost. For example, a five-state unordered character with states A through E has the character-state graph:

Character-state assignments are made to internal nodes of the tree so as to minimize the total number of character-state transformations (steps), using an algorithm based on that of Fitch (1971).

Note: *The "ordered" vs. "unordered" distinction does not apply to binary characters. The ordering of a character refers to the potential pathways of character transformation, and there is only one possible path between the two states of a binary character. Consequently, it makes no difference whether freely reversible binary characters are defined as "ordered" or "unordered."*

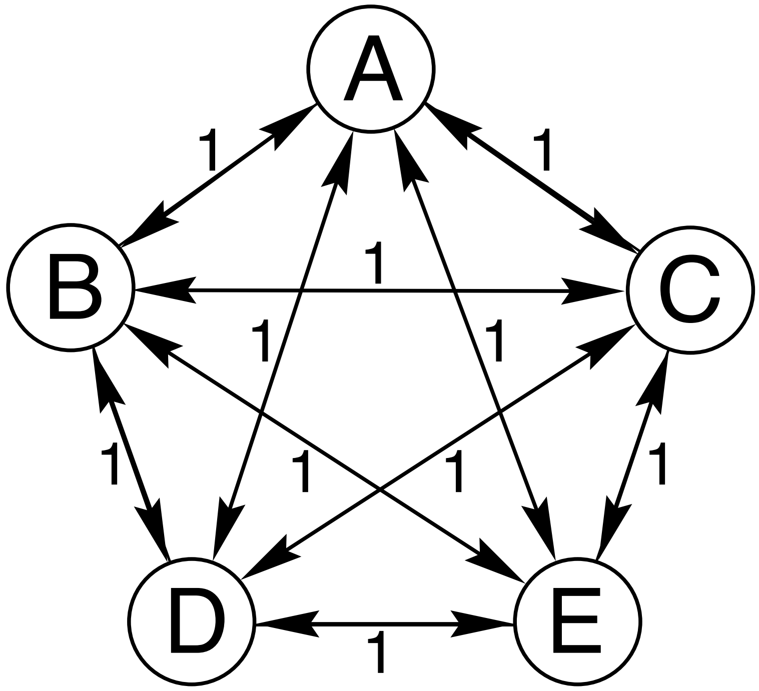


Figure 4.2 *Character-state graph for a 5-state unordered character.*

4.13.3 *Dollo Characters*

A ”Dollo” character is one that is consistent with the requirement that every derived character state be uniquely derived. If a hypothetical ancestor possessing the presumed ancstestral state for each character is included in the analysis, this definition corresponds to the traditional Dollo model in which each character state is allowed to originate only once during evolution and all homoplasy takes the form of reversals to a more ancestral condition (i.e., parallel gains of the derived condition are prohibited). As for ordered (Wagner) characters, character states are linearly ordered according to their position in the symbols list specified in the Format command of the Data block. However, we now define a ”forward transformation” as a change from a less derived state to a more derived state, and a ”backward transformation” as a change from a more derived state to a less derived state.

4.13.4 *Rooted vs. unrooted Dollo models*

PAUP* (as well as MacClade) differs from some other implementations of Dollo parsimony (e.g., Felsenstein’s PHYLIP) in that it can operate as an unrooted method. The only requirement is that the reconstructed character states be consistent with the constraint that each derived state be uniquely derived. Under this definition, the position of the root affects neither the assignment of character states nor the length of the tree. For example, both of the trees below, which differ only in the placement of the root, require two steps under the unrooted Dollo model, assuming that state 1 is the derived state:

That is, neither tree requires more than a single origination of state 1. (In the tree on the right, the derived state (1) is assumed to be ancestral with respect to the group ABCD, but derived relative to some more inclusive group.)

If, on the other hand, the trees are rooted by the attachment of a hypothetical ancestor possessing state 0, the left tree will be shorter (2 steps) than the right tree (3 steps):

The extra length in the right tree comes from the inclusion of the initial gain of state 1 in the tree length. This example makes it clear that trees computed under Dollo parsimony are intrinsically rooted only if we

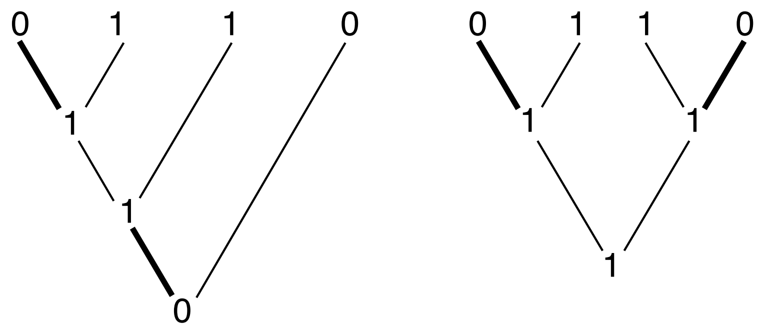


Figure 4.3 *Reconstructions for a Dollo character under two different rootings of an unrooted tree. Terminal taxa are labeled according to their state for the character in question.*

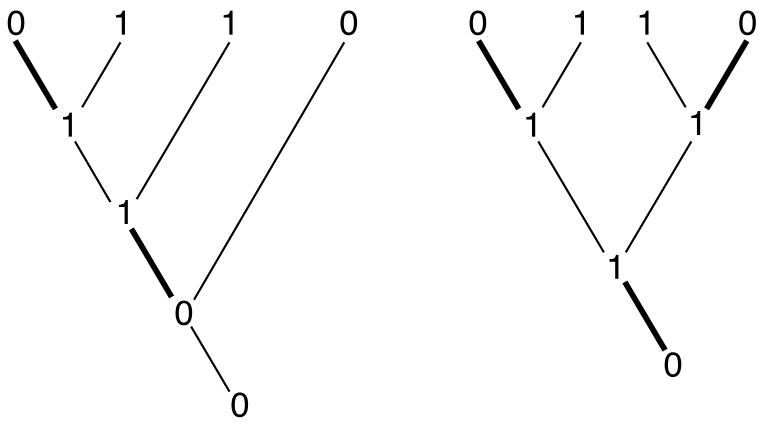


Figure 4.4 *Reconstructions for a Dollo character under two different rootings of an unrooted tree. Terminal taxa are labeled according to their state for the character in question.*

assume we know the state at the "outgroup node" [sensu Maddison et al. (1984)].

A more formal definition of the unrooted Dollo criterion is the following. A **character-state reconstruction** satisfies the Dollo constraint if it is possible to trace a path between any pair of nodes possessing the same character state without passing through a node possessing a less derived state. For example, reconstruction **A** below, while requiring fewer steps than reconstruction **B**, is not a Dollo reconstruction, as tracing the path connecting the two terminal nodes possessing state 1 requires passing through internal nodes possessing the less derived state 0. Reconstruction B, on the other hand, does satisfy the Dollo constraint.

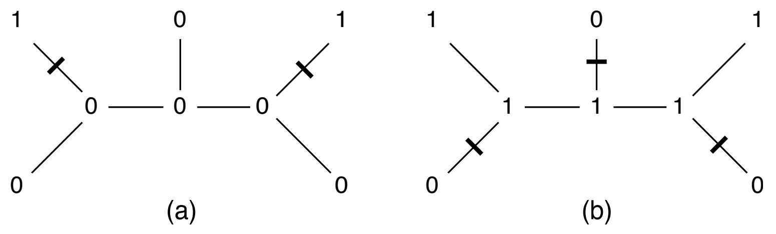


Figure 4.5 *Two reconstructions for a single character on an unrooted tree. Reconstruction a requires 2 steps but violates the Dollo constraint. Reconstruction b satisfies the Dollo constraint, but requires 3 steps.*

Note: Rooting the tree along any branch requires independent (parallel) gains of state 1 under reconstruction A, whereas all rootings of the tree are consistent with a unique origination of state 1 under reconstruction B, in accordance with traditional Dollo parsimony.

Unless an ancestor is included in the analysis-either explicitly or due to the presence of irreversible or asymmetric stepmatrix characters (see "Rooted vs. unrooted Dollo models")-PAUP* uses the unrooted Dollo method. If, on the other hand, an ancestral taxon is included, then PAUP* performs a rooted Dollo analysis. (If all characters are binary and are assigned polarity "up," as noted below, the rooted analysis corresponds to the implementation of Dollo parsimony in PHYLIP.) Since tree lengths will, in general, vary according to the position of the root, the result of the analysis is an intrinsically rooted tree-any specification of outgroups by the user is ignored. While the ability to obtain rooted trees without assuming an outgroup may seem appealing, it comes at a high price.

Suppose a derived character state appears both in our ingroup and in an undisputed outgroup taxon also included in the data matrix. The analysis will place a premium on making all of the taxa possessing the derived state a monophyletic group (subject, of course, to effects from other characters in the data set), since the alternative is likely to be many independent losses. As a result, taxa that would ordinarily be assigned to the ″outgroup″ may spring from within the ingroup. Even when no basis exists for identifying ″outgroup″ vs. ″ingroup″ taxa, the rooting is still likely to be more artifactual than meaningful. The assumption is that the ancestor of the full tree possesses the ancestral state for the character and that the derived states must evolve somewhere on the tree from this ancestral state. Thus, the tree tends to be rooted nearest the taxa that have the fewest derived states. This may in fact be what you want, but you should at least be aware of the reasons why the program places the root where it does.

To amplify on the points made in the preceding paragraph, note that Dollo parsimony is sometimes recommended for restriction site data (e.g., DeBry and Slade, 1985) because of the asymmetry in the probabilities of losing an existing restriction site vs. gaining a new site at a particular location. (If a site is present, any substitution at any position in the recognition sequence causes a site loss, whereas even if a particular sequence is one substitution away from being a site, exactly the right substitution at exactly the right position is required to convert the ″one-off site″ to a site.) Thus, a rooted Dollo analysis in which the ancestral state is assumed to be ″site absent″ will tend to root the resulting tree(s) near the taxa that have the fewest sites. Although one could argue that this approach to rooting is reasonable, it seems a bit arbitrary to me.

There are ways of circumventing the problems of using rooted Dollo parsimony for characters like restriction site data. One approach is to infer character polarity via traditional outgroup analysis and then use a mixture of Dollo and Camin-Sokal (irreversible characters, see below) parsimony. If the site occurs only in (some) members of the ingroup, then we would designate the ancestral state as ″absent″ and allow a single gain of the site followed by as many losses as would be required to explain the character (i.e., traditional Dollo parsimony). But if a site occurs in the ingroup and in the outgroup, the assumption that the site was already present in the common ancestor of the ingroup-plus-outgroup would be reasonable. In this case, we would designate the ancestral state for this character as ″present″ and then treat the character as an irreversible (rather than Dollo) character, allowing only losses of the site in accordance with the Dollo model. A somewhat simpler but logically equivalent approach is to constrain the ingroup to be monophyletic (either through the use of a heavily weighted "dummy" synapomorphy or by using the "topological constraints" feature of PAUP*) and use Dollo parsimony with an "all-absent" ("all-zero") ancestral taxon for all of the characters. Then if the site occurs in both the ingroup and the outgroup, a site gain will be forced along the basal branch of the tree (terminating at the common ancestor of the ingroup-plus-outgroup) and all subsequent character changes will be losses. If, on the other hand, the site occurs only within the ingroup, then a single origination will be assigned within the ingroup, perhaps with one or more subsequent losses.† Fortunately, the unrooted Dollo approach available in PAUP* and MacClade renders these more complicated approaches unnecessary, but users should understand the logical connections between the alternative methodologies.

Note: *Felsenstein (1984* ***CHECK MORE RECENT REF****) has described a variation on rooted Dollo analysis that he calls the "unordered Dollo" method. (This terminology is a bit unfortunate-"unpolarized" Dollo would have been a better name.) Unlike the Dollo method implemented in PAUP*, which assumes that the polarity is known, the unpolarized Dollo method evaluates each character on a given tree under all possible polarity assignments and chooses that polarity which allows the minimum number of changes. For example, in a two-state character with states 0 and 1, we would count first the number of changes required under the assumption that state 0 is ancestral, and then count the number of changes required under the assumption that state 1 is ancestral. Polarity is then assigned in accordance with the assignment requiring the fewest changes. I cannot imagine a situation in which one would be willing to assume that a character could not evolve from an ancestral state to a derived state more than once, but unwilling to postulate the character's polarity. Consequently, PAUP* does not implement this method. (If you can imagine such a situation, let me know.) The unpolarized Dollo approach is not appropriate for restriction site data, since allowing site presence to be ancestral and site absence to be derived would, under the Dollo model, imply that sites could only be lost once and then*

†An "all-missing" ancestor with polarity "up" [see "Direction of transformations (polarity)]" for all characters could also be used, with equivalent results. The same character state would be assigned to the node corresponding to the common ancestor of the ingroup-plus-outgroup in either case. The only difference is that with an all-absent (all-zero) ancestor, one step would be added to the tree length for every character in which the site was present in both the ingroup and outgroup, corresponding to the site gain along the basal branch. With an "all-missing" (unknown) ancestor, this additional step would not be required, because the change would be from the "missing" state to present (see "Missing Data"). Thus, the only difference between the two methods is that a constant is added to (or subtracted from) the tree length.

regained as many times as necessary, clearly an unreasonable assumption.

4.13.5 Polarity specification

For the unrooted Dollo method, PAUP* allows either the lowest or highest observed state (as defined by the Symbols list) to be the ancestral state (polarity ”up” vs. ”down,” respectively). If you do not explicitly specify ”up” or ”down,” ”up” is assumed. When an ancestor is included in the analysis (rooted Dollo), any state in the Symbols list may be designated as the ancestral state.

See ”Assigning Character Polarities” for information on how to designate character polarities for Dollo characters in PAUP*.

4.13.6 Irreversible Characters

Irreversible characters are equivalent to ordered characters with the additional constraint of irreversibility being imposed (Camin and Sokal, 1965). As for ordered (Wagner) characters, character states are linearly ordered according to their position in the Symbols list specified in the Format command of the Data block. However, we now prohibit transformations from a more derived state to a less derived state. For example, if `symbols="ABCDE"` and state A is defined as the ancestral state, the character-state graph is:

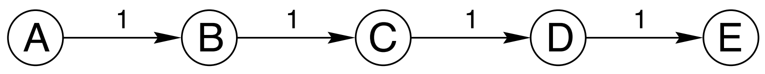


Figure 4.6 *Character-state graph for a 5-state irreversible character.*

4.13.7 Polarity specification

Although character transformations must always proceed in the direction of a more derived state, they need not proceed in a single direction with respect to the Symbols list. Any state in the Symbols list may be designated as the ancestral state; with states preceding and/or following this state representing more derived states. For instance, state C could instead have been chosen as the ancestral state in the example above. The character-state graph would then be:

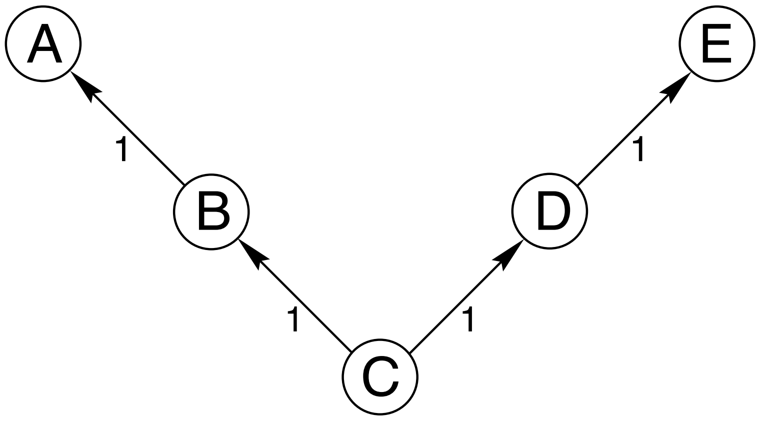


Figure 4.7 *Character-state graph for a 5-state irreversible character with ancestral state "C".*

See "Assigning Character Polarities" for information on how to designate character polarities for irreversible characters in PAUP*.

NOTE: *Felsenstein (1984* ***CHECK MORE RECENT REF****) has described a variation on irreversible-character parsimony that he calls the "unordered Camin-Sokal" method (analogous to his "unordered Dollo" method discussed above). Again, "unpolarized Camin-Sokal" would have been a more appropriate name. As in the unordered Dollo method, the unpolarized Camin-Sokal method evaluates each character on a given tree under all possible polarity assignments and chooses that polarity which allows the minimum number of changes. It seems exceedingly unlikely that one would be willing to assume that a character was irreversible, but ignorant as to its polarity, so I have not implemented unpolarized irreversibility in PAUP*.*

4.14 Defining Character Types

4.14.1 The ”Standard” Character Types

The standard character types are referred to in NEXUS and PAUP* commands as `ORD` (ordered), `UNORD` (unordered), `IRREV` (irreversible), and `DOLLO` (Dollo). Dollo and irreversible characters may have an optional suffix (`DOLLO.UP`, `DOLLO.DN`, `IRREV.UP`, `IRREV.DN`)

Table 4.2 List of Parsimony Variants Available in PAUP*

Parsimony Variants Implemented in PAUP*	Descriptive name	command-line syntax
Fitch (Fitch, 1971)	Unordered	`unord`
Wagner (Kluge and Farris, 1969; Farris, 1970)	Ordered	`ord`
Camin-Sokal (Camin and Sokal, 1965)	Irreversible	`irrev.up` and `irrev.dn`
Dollo (Farris, 1977)	Dollo	`dollo.up` and `dollo.dn`
Generalized (Sankoff and Rousseau, 1975)	User-defined	`usertype`

4.14.2 Assigning Character Polarities

Dollo and irreversible characters require the specification of character polarities (either implicitly or explicitly). For Dollo characters, there are three options for specifying the direction of ”forward” (less derived to more derived) vs. ”backward” (more derived to less derived) transformations. If the ancestor currently in effect has the ”missing” state, polarity can be either ”up” or ”down.” ”Up” specifies that states higher in the Symbols order are derived relative to states lower in the Symbols order (i.e., for `SYMBOLS=”01”`, state 0 is ancestral and state 1 is derived). ”Down” specifies the opposite: higher ordered states are ancestral to lower ordered states. If the state in the currently chosen ancestor is non-missing, this state defines the most ancestral state, with both lower and higher ordered states in the Symbols list being relatively more derived. Note that if the ’standard’ ancestor is in effect, this state will be ”missing,” and the ”up” vs. ”down” setting will apply.

Within a file or from the command line, these options are specified by optionally appending a suffix to the Dollo keyword. If no suffix is provided (`’DOLLO’`), then ”up” is assumed. ’`DOLLO.UP`’ and ’`DOLLO.DN`’ can be used to explicitly request the ″up″ and ″down″ options, respectively.

Likewise, for irreversible characters, there are three options for specifying the direction of allowed (less derived to more derived) transformations and disallowed (more derived to less derived) transformations. If the ancestral state defined by the current ancestor is not equal to ″missing,″ then this state represents the most ancestral state, with both lower and higher ordered states in the Symbols list being relatively more derived. If an ancestral state is ″missing,″ then the polarity may either be ″up″ or ″down.″ ″Up″ specifies that states higher in the symbols order are derived relative to states lower in the symbols order (i.e., for Symbols=″01″, state 0 is ancestral and state 1 is derived). ″Down″ specifies the opposite-higher ordered states are ancestral to lower ordered states. Note that if the ″standard″ ancestor is in effect, all ancestral states will be ″missing″, and the ″up″ vs. ″down″ setting determines the polarity.

Within a file or from the command line, these options are specified by optionally appending a suffix to the IRREV keyword. If no suffix is provided (′`IRREV`′), then ″up″ is assumed. ′`IRREV.UP`′ and ′`IRREV.DN`′ can be used to explicitly request the ″up″ and ″down″ options, respectively.

Menu equivalent: **Data > Set > Character Types**

4.14.3 Defining Your Own Character Types

In addition to providing the standard character types mentioned above, PAUP* allows you to define your own character types via the UseType command. Two kinds of user-defined character types are available. Character-state trees allow you to define a character-state graph that specifies a linear or branching relationship among the character states. Stepmatrices assign a cost for the transformation from every state to every other state.

The UseType command defines a character transformation type, as used in parsimony analysis to designate the cost of changes between states. UseType must be defined before they are referred to in any TypeSet. Character-type-description must follow the rules for character-state tree of stepmatrix description.

4.14.4 *Character-state trees*

Character-state trees are used to specify a non-linear, branching, relationship among character states, which are then otherwise treated as ordered characters. (Formally, a character-state tree imposes a ”partial order” on the character states.) In most earlier programs, users were required to recode character-state trees using additive coding (binary or otherwise; see below). PAUP* now provides this capability automatically.

As an example, suppose that you wish to assume that states *C* and *D* were derived independently from state *B*, which itself was derived from state A. The following character-state tree would be used to represent this relationship:

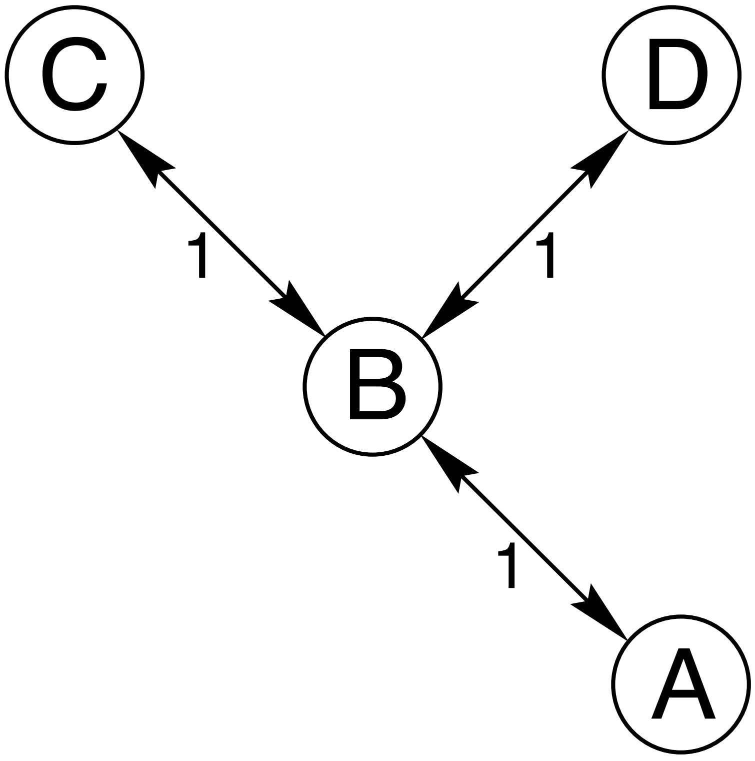


Figure 4.8 *A character-state tree for four states.*

Either of the following additive codings, breaking the character into a suite of independent characters, could be used in lieu of a character-state tree (see, e.g., **REFS**):

```
A  = 00 A = 000
B  = 10 B = 100
C  = 20 C = 110
D  = 11 D = 101
```

However, such a coding makes interpretation of the output more difficult because you would have to decode the new characters back to the original states of the character-state tree. In this case, the character-state tree specification

((C,D)B)A

provides an equivalent definition of the character, requires essentially identical computation time, and eliminates the encoding-decoding steps.

Characters defined as character-state trees are fully reversible, however they may evolve only along the paths specified by that definition. For instance, in the above example a change from state *C* to state *D* would imply passage through state *B* and would require two steps.

Any character state tree may also be defined as an equivalent stepmatrix character (see below). However, processing of explicitly defined character-state trees is *much* faster, so ordinarily you would want to avoid using stepmatrix character types to describe character-state trees. However, the use of stepmatrices does provide a way to assign different transformation costs to different branches of the character-state tree.

Finally, remember that character-state trees, like linearly ordered characters, are undirected. The character-state tree only specifies available paths for character-transformation; it does not imply anything about polarity. In the example above, states *B*, *C*, or *D* could just as easily have been placed at the base as shown below:

Character-state trees are described using a parenthetical notation that defines the shape of the tree. The nodes of the character-state tree are labeled by symbols corresponding to the character-state symbols used in the data matrix. Obviously, all of the states observed in the data matrix for a particular character must be represented in any character-state tree

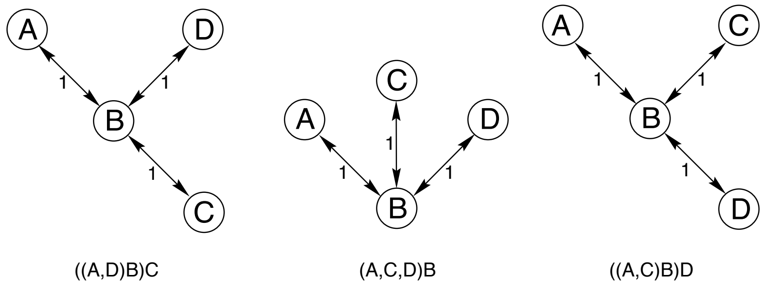


Figure 4.9 *Three rootings of a character-state tree that are equivalent to the one shown above.*

assigned to that character. Additional states that were not observed may also be included in the character-state tree as well.

The CSTree format is very similar to the Tree format in a Tree block. That is, character state trees are described in the parenthesis notation following the rules given for Trees of taxa. Instead of taxon labels, character state symbols are used. Thus

Command-line

- Type: `usertype cstree-name (cstree) = [(list of subtrees)][state-symbol]];`

where each subtree has the same format as the overall tree and the subtrees are separated by commas.

The system used to describe character-state trees is straightforward. Think of the character-state tree diagram as a set of roadways connecting the nodes of the tree (see figure below). The itinerary is to visit all of the nodes (character-states) of the character-state tree in a circuit beginning at the root node (state *D*), following two simple rules: (1) when you come to an intersection or fork in the road (internal node), always bear to the left, and (2) when you come to a dead end (terminal node), turn around. The path indicated by the arrows in the figure shows the sequence in which the nodes would be visited in this example.

To write the character-state tree description, perform the following operations as you make the circuit:

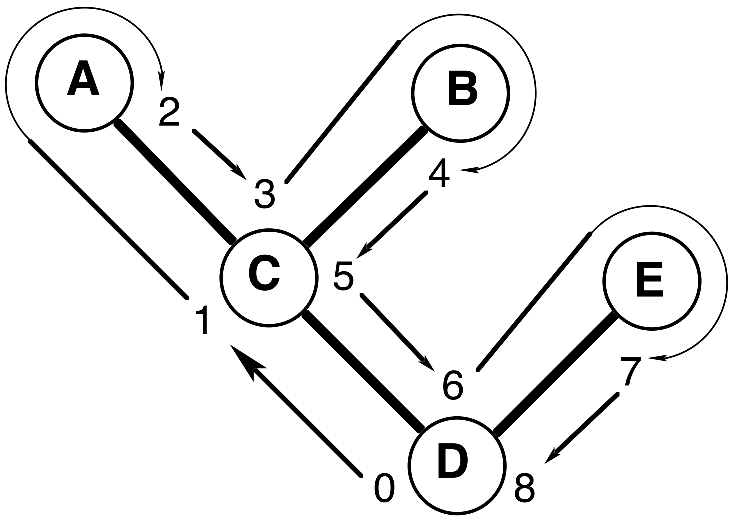


Figure 4.10 *A character-state tree and the circuit followed in writing its description.*

- When you leave a node traveling away from the root toward the leftmost descendant, write a left parenthesis.
- When you leave a node traveling away from the root toward any descendant other than the leftmost descendant, write a comma.
- When you leave the rightmost descendant of an internal node traveling toward the root, write a right parenthesis.
- When you visit a terminal node or visit an interior node for the last time, (optionally) write the symbol for the character state. If you omit the symbol for a state corresponding to an internal node of the character-state tree, PAUP* will use an asterisk to label that state in the output. For ease in interpreting the output, however, it is best to label all nodes on the character-state tree.

Applying these rules to the example above, the character-state tree description would develop as follows:

Number in sequence	Description to this point	Explanation
0	(	leaving for internal node D's left descendant

1 ((leaving for internal node C’s left descendant
2 ((A terminal node A visited
3 ((A, leaving for internal node C’s right descendant
4 ((A,B) terminal node B visited; leaving internal
    node C’s rightmost descendant
5 ((A,B)C internal node C visited for last time
6 ((A,B)C, leaving for internal node D’s last descendant
7 ((A,B)C,E) terminal node E visited; leaving internal
    node D’s rightmost descendant
8 ((A,B)C,E)D internal node D visited for last time

The character-state tree could therefore be defined using a UserType command in the Assumptions block as follows:

```
begin assumptions;
 usertype mycst cstree = ((A,B)C,E)D;
end;
```

The name `mycst` can be any name you choose. CSTree is required because, by default, user-defined character types are stepmatrices (see below) rather than character-state trees. Just for practice, here is a slightly more complicated example:

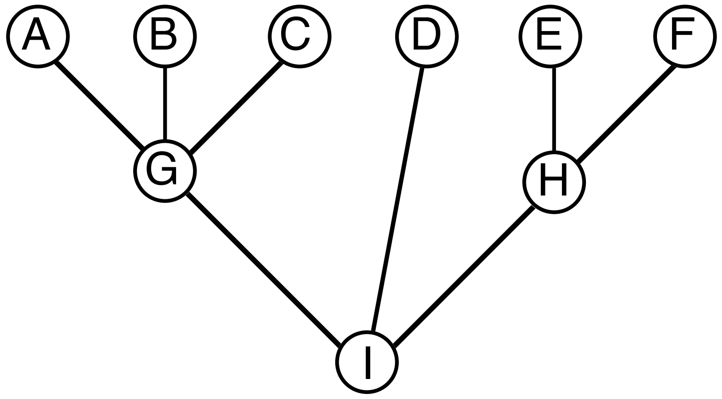


Figure 4.11 *Another character-state tree and its NEXUS-format description (below).*

NEXUS-format description of Figure 2.11.

((A,B,C)G,D,(E,F)H)I

In unusual situations, you may want to use character-state trees to define a linearly ordered (rather than branching) character, but with a different ordering from that implied by the Symbols list. For example, if Symbols = "012" and you want one character to be ordered as 0-2-1 rather than 0-1-2, you could declare a user-defined type with the character-state tree specification "((1)2)0". Other ordered characters would retain the 0-1-2 ordering.

You are not required to assign a character-state symbol to all internal nodes of the character-state tree. If you do not explicitly assign a symbol to an internal node, an asterisk (*) is used. For example, if none of the taxa in the data matrix actually possessed any of the states G, H, or I, the description of the character-state tree shown in the above tree could have been written as

```
((A,B,C),D,(E,F));
```

The states corresponding to the internal nodes of the character-state tree would then be shown as an asterisk in the output. This practice is not recommended, however, as it then becomes impossible to distinguish between different nodes of the character-state tree that are all represented by the same symbol in the output.

User-defined character-state trees cannot include multistate taxa. In that case you must code the character-state tree using a stepmatrix (see below).

4.14.5 Stepmatrices

A stepmatrix is a square matrix specifying the distance from every character state to every other state. These distances represent the "cost" in tree-length units of the corresponding transformations. (Transformations may be completely forbidden by coding 'i', for infinity, as the transformation cost.) For each stepmatrix character, PAUP* uses dynamic programming algorithms (Sankoff and Rousseau, 1975; Sankoff and Cedergren, 1983) to determine the minimum possible length on each tree it evaluates and to reconstruct hypothetical ancestors consistent with this length.

Stepmatrices may be either **symmetric** or **asymmetric**, corresponding to the usual definition of matrix symmetry (i.e., a cost matrix **C** is symmetric if $c_{ij} = c_{ji}$ for all *i,j*). Symmetric stepmatrices imply free reversibility of characters, because the cost of reversing a given transformation is equal to the cost of the original transformation.

NOTE: *Asymmetric stepmatrices force a rooted tree. If the stepmatrix is asymmetrical, the taxon designated "ancestor" is very important. If you want to have asymmetrical transition penalties but remain agnostic regarding the ancestral condition of the characters, you should include an ancestor with all-missing values. See the section "Character Types" for the relationship between character type and tree rooting.*

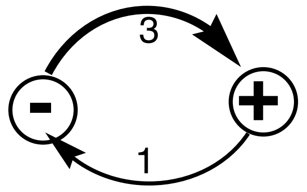
The "generalized parsimony" approach provided by stepmatrix characters is extremely powerful. Researchers with nucleotide sequence data can assign different weights to transitions vs. transversions (and even to different kinds of transitions and transversions):

$$\begin{tikzcd}
A \arrow[r, leftrightarrow, "1"] \arrow[rr, leftrightarrow, bend right, "5"'] \arrow[rrr, leftrightarrow, bend right=50, "5"'] & G \arrow[r, leftrightarrow, "5"] \arrow[rr, leftrightarrow, bend left, "5"] & C \arrow[r, leftrightarrow, "1"] & T
\end{tikzcd}$$

$$\begin{array}{ccccc}
 & A & C & G & T \\
A & 0 & 5 & 1 & 5 \\
C & 5 & 0 & 5 & 1 \\
G & 1 & 5 & 0 & 5 \\
T & 5 & 1 & 5 & 0
\end{array}$$

Figure 4.12 *Character-state graph and corresponding stepmatrix for a character type that assigns five times more weight to transversions than to transitions.*

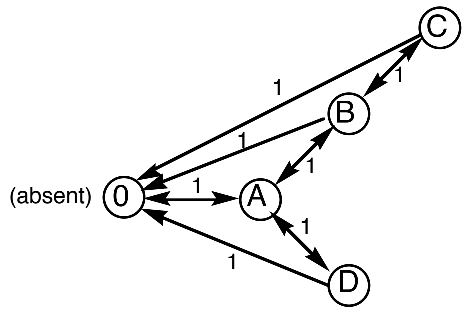
For restriction map data, site gains can be given higher weight than site losses, so that parallel loss and gain-loss events are preferred over parallel gains and loss-regains (Templeton, 1983b,a), avoiding the perhaps overly severe strict prohibitions imposed by the Dollo and Camin-Sokal models (DeBry and Slade, 1985; Swofford and Olsen, 1990):



$$\begin{array}{c|cc} & - & + \\ - & 0 & 3 \\ + & 1 & 0 \end{array}$$

Figure 4.13 *Character-state graph and corresponding stepmatrix for a character type that assigns three times as much weight to a gain than a loss.*

Stepmatrices may be used to define models of character transformation that cannot be expressed under any other available coding method. For instance, morphologists can define "partially unordered" characters in which some transformations are required to follow a specified order but others may occur freely. In the example below, a character-state tree for several alternative "present" states is specified, but a transformation to the "absent" condition is permitted to occur with equal cost from any state:

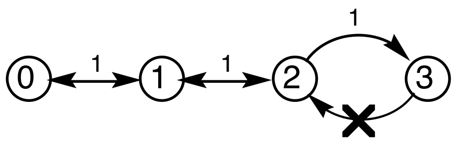


$$\begin{array}{c|ccccc} & 0 & A & B & C & D \\ 0 & 0 & 1 & 2 & 3 & 2 \\ A & 1 & 0 & 1 & 2 & 1 \\ B & 1 & 1 & 0 & 1 & 2 \\ C & 1 & 2 & 1 & 0 & 3 \\ D & 1 & 1 & 2 & 3 & 0 \end{array}$$

Figure 4.14 *Character-state graph and associated stepmatrix for a "partially unordered" character. Transformations between states A, B, C, and D follow a character-state tree, but immediate losses may occur from any state.*

"Partial irreversibility," where reversals are permitted for some transformations but not for others, can also be implemented using stepmatrices:

Most of the other standard types may be defined equivalently as stepmatrices. For example, the stepmatrices



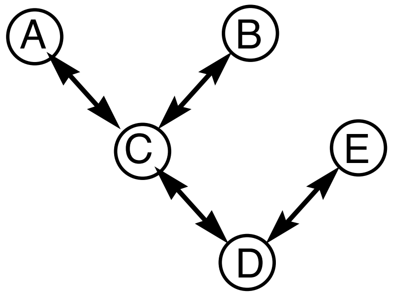
```
   0 1 2 3
0  0 1 2 3
1  1 0 1 2
2  2 1 0 1
3  i i i 0
```

Figure 4.15 *Character-state graph and associated stepmatrix for a "partially irreversible" character. State "3" cannot reverse to any other state, hence the entries of infinity ("i") in the last row of the stepmatrix.*

```
0 1 2     0 1 1         0 1 2
1 0 1   , 1 0 1   , and i 0 1
2 1 0     1 1 0         i i 0
```

correspond to the ordered, unordered, and irreversible types, respectively, for a character with 3 or fewer states. However, stepmatrix characters require significantly more computation than other character types and should be used only if it is not possible to use one of the standard types.

Stepmatrices may also be used to describe character-state trees. For example, the character-state tree



= ((A,B)C,E)D

Figure 4.16 .

can be described using the stepmatrix:

```
   A  B  C  D  E
A  -  2  1  2  3
```

```
B  2  -  1  2  3
C  1  1  -  1  2
D  2  2  1  -  1
E  3  3  2  1  -
```

However, there are enormous computational advantages to using an explicitly defined character-state tree rather than an equivalent stepmatrix. Again, use stepmatrices only when another equivalent type is unavailable.

See ”The Assumptions block” for details on how to define a stepmatrix in an input file.

It is possible to define stepmatrix characters that violate the ″triangle inequality.″ This inequality is a property of distances in Euclidean space, where the length of one side of a triangle is always less than the sum of the lengths of the other two sides. This means the shortest distance between two points will always be a straight line. If a stepmatrix is defined that violates this rule, one side is actually longer than the sum of the other two sides. For example, the following triplet violates the inequality, as the distance (*b*,*c*) is longer than the sum of the distances (*a*,*b*) and (*a*,*c*).

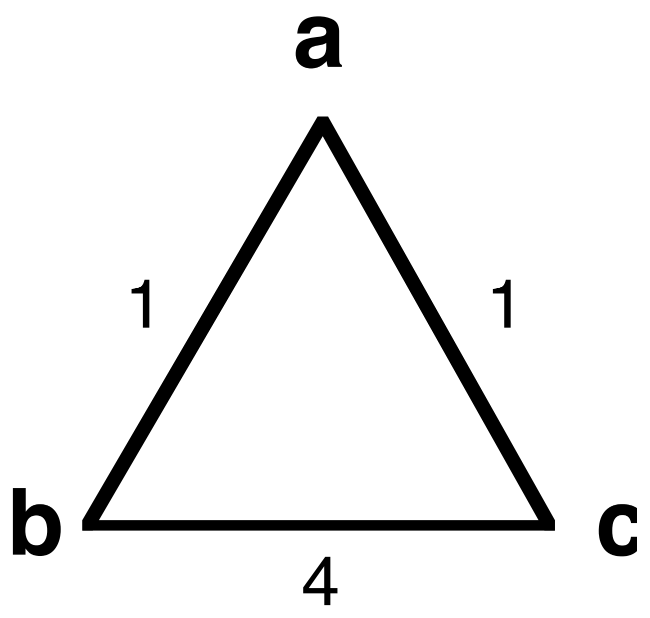


Figure 4.17 .

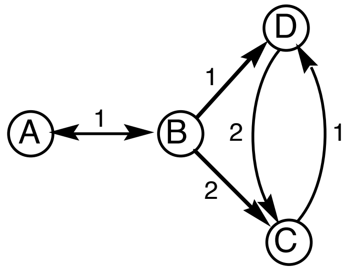
This would result from the following stepmatrix:

	a	b	c
a	-	1	1
b	1	-	4
c	1	1	-

If a stepmatrix contains a triplet that violates the ″triangle inequality,″

PAUP* will display a warning the stepmatrix is internally inconsistent, but will still allow it to be used in an analysis. In that case, you must decide if you really want to disallow shortcuts such as $b \rightarrow c$, and what the biological meaning of that stepmatrix really is. The implications of having stepmatrices which violate the triangle inequality are discussed in more detail in Maddison and Maddison (1992).

To define a stepmatrix character, first draw the character-state graph corresponding to the assumptions that you wish to impose for the character type. The character-state graph should have arrows connecting all allowed transformations (this is where you can incorporate the changes associated with a multistate taxon) with a cost (weight) associated with each transformation. Then, assign each element s_{ij} of the stepmatrix according to the sum of the costs of all the transformations required to convert state *i* (stepmatrix row) to state *j* (stepmatrix column). If there is more than one path from state *i* to state *j* (i.e., there are one or more cycles in the character-state graph), choose the path the allows the smallest total cost. Here is one more example, intentionally made rather unusual in order to illustrate a few points:



$$\begin{array}{ccccc} & A & B & C & D \\ A & 0 & 1 & 3 & 2 \\ B & 1 & 0 & 2 & 1 \\ C & i & i & 0 & 1 \\ D & i & i & 2 & 0 \end{array}$$

Figure 4.18 *A character-state graph and its associated stepmatrix.*

First, note that state *C* can only transform into state *D* and vice versa, since the *B*-to-*D* and *B*-to-*C* arrows are unidirectional. Hence, the matrix entries for *C* and *D* to *A* and *B* are "infinity," represented by the character 'i' in the stepmatrix definition. Also observe, for example, that there are two paths from *A* to *C*: $A \rightarrow B \rightarrow C$ (3 steps) and $A \rightarrow B \rightarrow D \rightarrow C$ (4 steps). Therefore, we let $s_{A,C} = 3$. The stepmatrix could be defined in an ASSUMPTIONS block as follows:

```
   begin assumptions;
```

```
    usertype weird stepmatrix = 4 ABCD
     0 1 3 2
     1 0 2 1
     i i 0 1
     i i 2 0
     ;
   end;
```

The name `weird` can be any name you choose; 4 is the number of rows and columns in the stepmatrix; and `ABCD` is the list of states corresponding to these rows and columns. Also, the Stepmatrix keyword is not actually required, as this is the default user-defined type class.

The Stepmatrix format is as follows:

```
usertype mymatrix (stepmatrix)=n
         [s  s  s  s]
         .   k  k  k
         k   .  k  k
         k   k  .  k
         k   k  k  . ;
```

where n is the number of rows and columns in the step matrix, the s's are state symbols, and the k's are the cost for going between states. n can take any value ≥ 2. Diagonal elements may be listed as periods. If a change is to be prohibited, then one enters an "i" for infinity. Typically, the state symbols will be in sequence, but they need not be. The following matrices assign values identically:

```
usertype mymatrix (stepmatrix)=4
         [0  1   2  3]
         .   1   5  1
         1    .  5  1
         5   5   .  5
         1   1   5  .;
usertype mymatrix2 (stepmatrix)=4
          [2  0  3  1]
          .   5  5  5
          5   .  1  1
          5   1  .  1
          5   1  1  .;
```

NOTE: *The diagonal elements of a stepmatrix are always set to zero in PAUP*, so you can put any one-character symbol you want there (e.g., '.' or '-') to improve readability. This is an extension to the NEXUS format, however, so other programs may not deal correctly with nonzero entries on the diagonal. MacClade, for example, allows you to substitute periods or hyphens for zeros, but other characters will generate an error.*

Remember that defining a character type does not automatically assign that type to the characters in the data matrix. Before a user-defined character type will have any effect on subsequent analyses, you must assign the type to one or more characters using one of the methods described in the section below.

4.14.6 Verifying USERTYPE definitions

Use the ShowUserType command to show all user-defined character types. There are no options for the command.

Command-line

- Type: `Showusertype` [*user-type-name*];

Menu equivalent: **Data > Show Other > UserTypes**

You can use the ShowUserType command to verify that you have defined your character types in the way you intended. The resulting output will have one of the following two forms:

```
Character-state tree 'one':

   *--e--c--a
   |  |  |
   |  |  b
   |  |
   |  d
   |
   j--h--f
   |  |
   |  g
   |
   i
```

```
Stepmatrix '10_1':


            TO: a       c       g       t
  FROM: a       -      10       1      10
        c      10       -      10       1
        g       1      10       -      10
        t      10       1      10       -
```

Because of the limitations of lineprinter-style graphics, this ouput for character-state trees can be confusing.

Example:

Command-line

- Type: `usertype a cstree=(a,b,c); showuser;`

generates the following ouput:

```
*--a
|
+--b
|
c
```

At first glance this implies a bifurcating cstree, which is not what was specified in the original usertype command. The trick is to ignore all apparent nodes that do not have either a state symbol or an asterisk (*). Thus, a, b, and c all connect directly to *, because the apparent node joining the edge leading to b and c does not really exist.

4.14.7 *Assigning Character Types*

There are four ways to specify character types:

- By using the DefType option in an Assumptions block. For example, the following Assumptions block sets the character type for all characters to ”ordered”:

```
begin assumptions;
 options deftype=ord;
end;
```

Note that the default character type must be one of the predefined character types; it can not be a user-defined type.

- By defining a type-set using one or more TypeSet commands in the Assumptions block. For example, to make characters 3 and 7 unordered, characters 9 through 13 Dollo, and the remaining characters ordered, you could use the following Assumptions block:

```
begin assumptions;
 options deftype=ord;
 typeset *mytypes = unord:3 7, dollo:9-13;
end;
```

The asterisk preceding `mytypes` is important. It informs PAUP* that you want the type-set to go into effect immediately. Otherwise, the type-set would not be used unless it was specifically invoked by an Assume command or a **Data > Choose Assumption Sets**

menu command.

NOTE: Any characters not explicitly assigned a type in a TypeSet command use the character-type specified by the DefType option in the Options command. If you do not use an ”OPTIONS DEFTYPE=___” directive, the default character type is Unord.

- By using the Ctype command (either from within a PAUP* block or from the command-line). The same character types assigned in the example above could be specified using the command:

Example:

Command-line

- Type: `ctype ord:all, unord:3 7, dollo:9-13;`

Menu equivalent

- Select **Data > Set Character Types...**

  click **All**, then **Ordered**.
- Select characters 3 and 7, then click **Unordered**. Finally, select characters 9 through 13 and select **Dollo**, and select **Done**

NOTE: The type-name must be one of the standard character types (ORD, UNORD, DOLLO, DOLLO.UP, DOLLO.DN, IRREV, IRREV.UP, or IRREV.DN) or the name of a user-defined character type (see, USERTYPE"). Each character-list consists of one or more character number, character name (see, "CHARLABELS"), or character set name (see, "CHARSET"). The characters specified by character-list are assigned the immediately preceding character-type. Any number of character-type:character-list pairs, separated by commas, may be specified. Later specifications override earlier specifications in a CTYPE command, so that characters 3, 7, and 9-13 are set to UNORD or DOLLO as intended, despite the preceding ORD ALL. This sometimes saves effort, since the alternative is to specify every character type explicitly.

Example:

Command-line

- Type: `ctype ord:1 2 4-6 8 14-.,unord:3 7,dollo:9-13;`

Menu equivalent

- Select **Data > Set > Character Types...**
- Click **All**, then **Ordered**.
- Select characters 3 and 7, then click **Unordered**. Finally, selected characters 9 through 13 and select **Dollo**, and select **Done**

4.14.8 Defining Ancestral States

Ancestral states are used in PAUP* for three purposes. First, some character types, including irreversible and asymmetric stepmatrix characters, automatically force a hypothetical ancestor to be included in the analysis; the ancestral states assign a state to this ancestor for each character. Second, even when it is not required, you may choose to include an ancestral taxon in a search (i.e., to explicitly specify character polarities). In this case, the ancestor is treated as an additional taxon, and trees computed during the search are automatically rooted at the point where the ancestor connects to the tree. Third, if you input user-defined trees as rooted trees (i.e., you use the Tree rather than the Utree command), character states must be assigned to the ancestor of the full tree.

Ancestral states are defined by an Ancstates command in the Assumptions block. In addition to user-defined Ancstates settings, a ″standard″ Ancstates definition is defined to have the ″missing″ state for all characters. If no other Ancstates definition appears in the Assumptions block, the default ancestor will be the ″standard″ (all-missing) one.

For example, to define an ancestor that assigns state 0 for all characters to the ancestor named ″allzero,″ you would use a command such as:

```
ancstates allzero = 0:all;
```

Of course, you do not have to assign the same state for all characters. For example, the command

```
ancstates mixed = 0:1 3 6-10, 1:2 4 12;
```

assigns to the ancestor named ″mixed ″ state 0 for the first, third, and sixth through tenth characters and state 1 for the second fourth, and twelfth characters. Any character numbers not explicitly assigned a character state (e.g., characters 5 and 11 above) retain the default ″missing″ state.

If two different states are assigned for the same character in an Ancstates command, the last assignment takes priority. This is convenient when you want to assign a state other than ″missing″ to most of the characters and a different non-missing state to a few characters. For example, the command

```
ancstates mostly_a = a:all, b:2 5;
```

assigns state ′a′ to ancestor ″mostly_a″ for all characters other than 2 and 5, which get state ′b′.

Instead of using ranges of characters as above, you can also explicitly specify each ancestral character state by using the Vector format option. This is useful if not necessary if the character states for the ancestor are very heterogeneous. For example, an ancestor with five characters would be specified by the following Vector statement, where name is the name of the Ancstates set.

```
ANCSTATES name VECTOR = 0 1 2 0 2;
```

Any number of ancestors may be defined (via multiple Ancstates commands), but only one is in effect at any one time. You may select the current ancestor in any of three ways:

- By preceding the name in an Ancstates command with an asterisk. For example, the commands

```
ancstates allzero = 0:all;
ancstates *allone = 1:all;
ancstates mixed = 0:1-10 1:11-20;
```

define three ancestors, with ancestor ”allone” being the currently chosen ancestor.

- By using the Assume command. For example, the command

```
assume ancstates = allzero;
```

would make ancestor ”allzero” the current ancestor.

- By using the **Data > Choose Assumption Sets**

  menu command.

Ancstates commands are typically issued from within an Assumptions block, however you may also issue them from within a PAUP* block or via the command line.

4.14.9 *Displaying ancestral states*

Use the ShowAnc command to request a listing of the ancestral character-states currently in effect. No options are available.

Syntax: `ShowAnc;`

Menu equivalent: **Data > Show Other > AncStates**

This ouputs the name of the current ancestor and its states. For example:

```
Character states for current ancestor: ”new”


                          1 1 1 1 1 1 1
           1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6
------------------------------------------
   Zapped:             *
   Uninf.:                       * *
 Excluded:       * *
------------------------------------------
new        ? ? ? ? ? ? ? 0 0 0 0 0 0 0 0 0
```

The status lines are omitted if Set Nocmstatus is used, or this output is suppressed in the **Options > Character Matrix Format**

dialog box.

5 Tree Searching and Reconstruction

5.1 Introduction

For many readers, this chapter may be of the most interest. The options available to search for trees PAUP* are to numerous to give examples for everything. The reader should spend time exploring the various options available for tree searching. PAUP* provides two basic classes of methods for searching for optimal trees, exact methods and heuristics. Exact methods guarantee to find the optimal tree(s) but may require a prohibitive amount of computer time for medium- to large-sized data sets. Heuristic methods do not guarantee optimality but generally require far less computer time. Either exact or heuristic methods may be employed by the bootstrap. The options available to each type of search are discussed under the appropriate heading below, but some options are available for all types of searches.

Although searches are designed to find optimal trees, you can request that PAUP* retain near-optimal trees as well. If you wish to do so, you can specify a tree length below which trees should be kept. Once you have done so, the optimal and near-optimal trees in memory can be saved or manipulated further. Next, you may choose during any search to collapse any zero-length branches. If you choose not to do so, all resolutions of optimal trees will be retained, even if there is no evidence for some of them. Next, you may choose to enforce topological constraints during the search (see the section “Searching under topological constraints”), either keeping or discarding trees that are compatible with the constraint tree.

5.2 Tree Terminology

The most general terminology for describing the various components of trees is derived from the field of graph theory [e.g., Harary (1969); Gould (1988)]. Some of this terminology is reviewed briefly here, although formalism is minimized. A **graph** consists of a set of **vertices** and a set of **edges**, where each edge is a line joining a pair of vertices. Two distinct vertices are adjacent if they are joined by an edge; the edge is said to be **incident to** those vertices. The **degree** of a vertex is the number of edges with which the vertex is incident. A **path** is a sequence of distinct edges such that each edge shares one vertex in common with the preceding edge in the sequence and the other vertex in common with the next edge in the sequence. A path connecting a pair of vertices can also be described as a sequence of vertices, with each vertex adjacent to the preceding vertex in the sequence. A graph is **connected** if there is at least one path from any vertex to any other vertex. A **cycle** is a path that connects a vertex to itself in which no vertex is repeated. Most importantly for our purposes here, a **tree** is a connected acyclic graph.

Vertices and edges on trees are often called **nodes** and **branches**, respectively, and this terminology will be used throughout the manual. Nodes are called **terminal nodes** if they have degree one and **internal nodes** otherwise. Unfortunately, the above terminology is not standardized, and much synonymy exists. Terminal nodes are also called **tips** or **leaves**; branches (edges) are also called **links**, **segments**, **intervals**, or **internodes**. Terminal nodes corresponding to biological taxa are often called **Operational Taxonomic Units** (OTUs) or simply **terminal taxa**. Similarly, internal nodes are sometimes referred to as **Hypothetical Taxonomic Units** (HTUs) or **hypothetical ancestors**.

A tree is **binary** if none of its internal nodes has degree exceeding three. If a binary tree has at most one vertex of degree two, it is also **full**. Full binary trees are sometimes called **strictly bifurcating** or **fully dichotomous** trees. A node having degree greater than three (e.g., one immediate ancestor and three immediate descendants; see below) corresponds to a **polytomy** or a **multifurcation**, and trees containing one or more polytomies are sometimes called **polytomous** (nonbinary) trees.

A tree is **rooted** if there is a special node (the root) that imparts a direction upon the tree such that nodes lying on a path connecting the root to any other node are **ancestors** of (ancestral to) nodes in the path that are further from the root and **descendants** of (descendant to) nodes closer to the root. Typically, the root is an internal node having degree two, however in PAUP* the root is usually considered to be an additional terminal node positioned at the base of the tree. A **subtree** is a connected subset of a tree.

A subtree consisting of all of the nodes and branches that descend from a particular internal node *v* on a rooted tree is called the "subtree rooted at *v*" or simply "*v*'s subtree." Phylogeneticists often refer to the subtree rooted at an internal node as a **clade**.

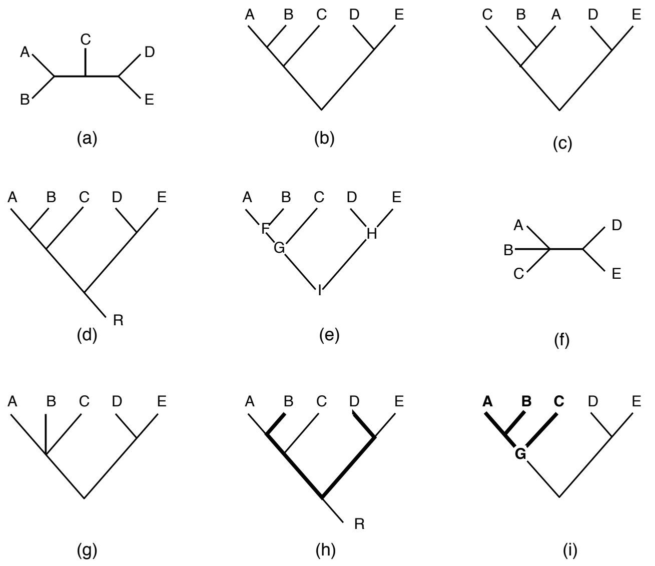


Figure 5.1 *Tree definitions. (a) Unrooted binary tree. (b) Rooted binary tree (rooted at an internal node of degree 2). (c) Another rooted binary tree. Trees b and c are equivalent unordered trees but distinct ordered trees. (d) Binary tree rooted at terminal taxon R. (e) Completely labeled rooted binary tree. (f) Unrooted nonbinary tree. (g) Rooted nonbinary tree. (h) Path between terminal nodes B and D (heavy lines). (i) Subtree of node G (heavy lines and boldface).*

A tree is **ordered** if the branches incident to each node are assigned in some nonarbitrary (fixed) order. If the order in which the branches are connected is irrelevant or arbitrary (as is generally the case for phylogenetic trees), then the tree is said to be **unordered**. (Although the trees computed by PAUP* are unordered in the sense that free rotation of subtrees around internal nodes does not affect the relationships implied for terminal taxa, an order may be imposed based on other criteria purely for output purposes.) Trees are also classified according to the way in which nodes are labeled. In phylogenetic analysis, trees are usually considered to be **terminally labeled**. That is, the terminal nodes correspond exactly to the biological taxa under study, but the labeling of the internal nodes is arbitrary. If all internal nodes are associated with actual objects (e.g., fossil taxa) and are not merely hypothetical constructs, the trees are **completely labeled**. Note that although some programs (e.g., MacClade) may permit actual taxa to occupy ancestral positions, PAUP* considers only terminally labeled trees; if a taxon is assigned to an internal node (e.g., in a user-specified tree description) it is "popped out" to a terminal position.

Examples of the types of trees discussed above are shown below.

5.2.1 *Number of possible bifurcating trees*

The total number of distinct, unrooted, terminally labeled, strictly bifurcating, trees for T terminal taxa is given by the formula

$$B(T) = \prod_{i=3}^{T}(2i-5) \tag{5.1}$$

(Cavalli-Sforza and Edwards, 1967; Felsenstein, 1978b). Table 2 shows the value of $B(T)$ for several values of T. Obviously, the number of trees quickly becomes quite large as the number of taxa increases

Table 2. The number of unrooted, binary, terminally labeled trees, B(T), for T terminal taxa.

T	B(T)
3	1
4	3
5	15
6	105
7	945
8	10,395
9	135,135
10	2 x 10^6
15	8 x 10^{12}
20	2 x 10^{20}
50	3 x 10^{74}

5.3 Setting an optimality criterion

PAUP* searches for optimal trees under three different optimality criteria (parsimony, likelihood, and distance). Although the methods used to search for trees does not vary across the criteria, different assumptions go into selecting a tree that is "optimal" under each of the three criteria (see individual chapters on parsimony, likelihood, and distance for more details).

Before performing a search, you want to chose an optimality criteria. In order to select the likelihood criterion you would:

Setting Optimality Criterion:

- Select **Analysis > Likelihood**

5.4 Exact Methods

5.4.1 Exhaustive search

The conceptually simplest approach to the search for optimal trees is simply to evaluate every possible tree. Assuming that exact methods exist for evaluating the length of any particular tree, an algorithm that generates all possible tree topologies and evaluates each one is guaranteed to find all of the optimal trees. The algorithm outlined in the figure below can be used for this purpose. Initially, the first three taxa in the data set are connected to form the only possible unrooted tree for these taxa (row 1). In the next step, the fourth taxon is added to each of the three branches of the three-taxon tree, thereby generating all three possible unrooted trees for the first four taxa (row 2). The process continues in a similar fashion, adding the ith taxon to each branch of every tree (containing $i - 1$ taxa) generated during a previous step. Thus, for example, row 3 contains all 15 possible trees for the first five taxa, obtained by adding the fifth taxon to each of the five possible branches for the three trees obtained at the four-taxon stage. This process demonstrates the rationale for equation 5.1 for counting the number of possible unrooted bifurcating trees for T taxa: for each of the possible trees for $i - 1$ taxa, there are $2(i - 1) - 3 = 2i - 5$ branches to which the ith taxon can be connected. Note that the order of addition is immaterial; one could just as well have chosen a taxon at random to connect to the tree at each step.

Evaluation of Equation 5.1 (see Table 2) quickly reveals why exhaustive search procedures are useful only for small numbers of taxa. There are over 2 million trees for 10 taxa and 34 million trees for 11 taxa, so it is doubtful that exhaustive search strategies will be useful beyond 11 taxa.

Exhaustive methods are not generally useful for more than 10 or 11 taxa, since there are over 2 million trees for 10 taxa and 34 million trees for 11 taxa. Computation of all trees for larger numbers of taxa is impractical at present. With very large data sets, heuristic methods may be the only avenue available for analysis. If however you do have a data set small enough for exhaustive analysis, there are several useful options besides the standard search options (see the section above on options affecting more all search algorithms). Obviously, addition sequence is irrelevant in an exhaustive search, since all trees are going to be evaluated anyway. The most useful option is the ability to calculate a frequency distribution of tree lengths. Normally, obtaining the frequency distribution is the only reason for doing an exhaustive rather than branch-and-bound search, so you will probably always specify this option.

For the following exercises, let us use the primate example dataset. Since the dataset 12 taxa, we will speed things up a bit by deleting one taxon (*Lemur catta*)

Delete Lemur:

- Select **Data > Delete/Restore taxa...**
- Select "Lemur_catta" under **Nondeleted taxa:** Box and click **Delete**
- Now click **OK**

Example 1:

Exhaustive Search:

- Select **Analysis > Exhaustive search...**
- Make sure the **Tree-score frequency distribution** dialog is checked
- Select the **Histogram with** option and enter **5 classses**

This will begin an exhaustive search and use an interval of five steps in the tree-length histogram. If you wish to save the interval distribution to a text file for input to other programs you must use the SaveFD option or the **Save tree-length frequencies to file** selection in the **Exhaustive** (search) dialog box. The SaveFD option is not persistent. You must also specify a filename to receive the output, either by using the File option or by specifying a filename in the dialog box that appears when the save option

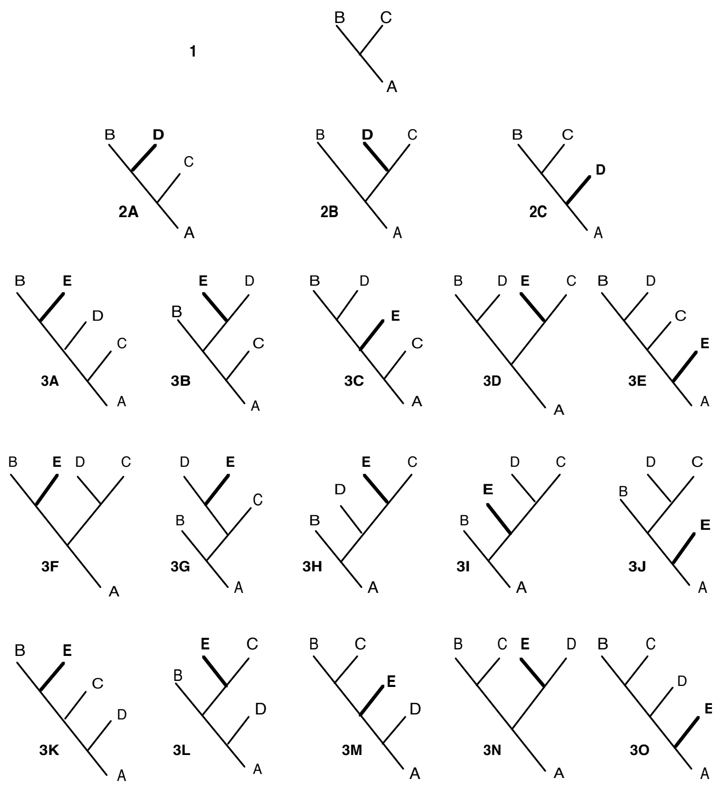


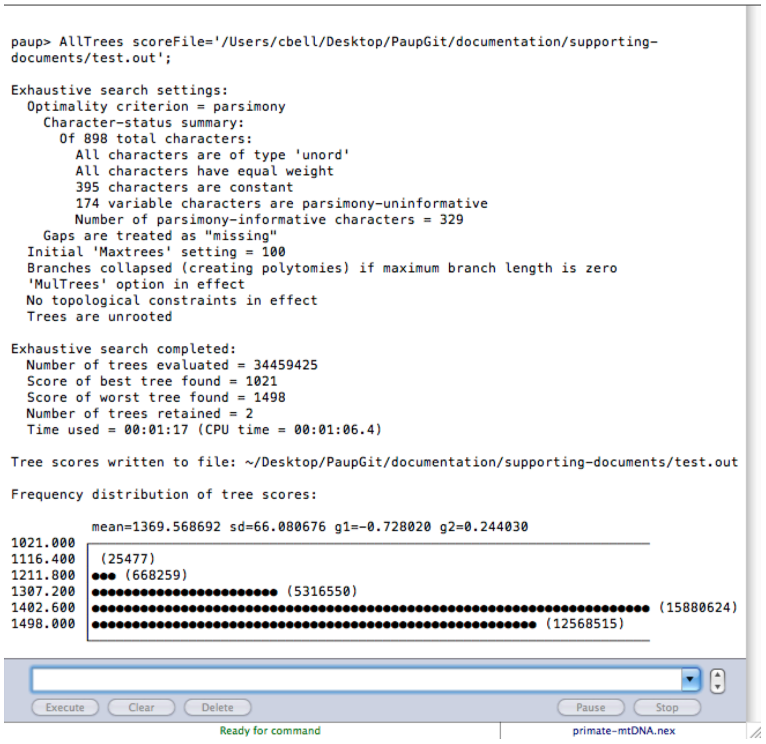
Figure 5.2 *Generation of all possible binary tree topologies for five terminal taxa.*

is selected. For example, the following command will begin an exhaustive search, use an interval of five steps on the histogram, save the distribution to the file "test.out", replacing it without prompting if it already exists.

Example 2:

Exhaustive Search:

- Select **Trees > Exhaustive search...**
- Make sure the **Tree-score frequency distribution** dialog is checked
- Select the **Histogram with** option and enter **5 classses**
- Finally, check the **Save tree scores to file** and Type: test.out and click **Save**



```
paup> AllTrees scoreFile='/Users/cbell/Desktop/PaupGit/documentation/supporting-
documents/test.out';

Exhaustive search settings:
  Optimality criterion = parsimony
    Character-status summary:
      Of 898 total characters:
        All characters are of type 'unord'
        All characters have equal weight
        395 characters are constant
        174 variable characters are parsimony-uninformative
        Number of parsimony-informative characters = 329
    Gaps are treated as "missing"
  Initial 'Maxtrees' setting = 100
  Branches collapsed (creating polytomies) if maximum branch length is zero
  'MulTrees' option in effect
  No topological constraints in effect
  Trees are unrooted

Exhaustive search completed:
  Number of trees evaluated = 34459425
  Score of best tree found = 1021
  Score of worst tree found = 1498
  Number of trees retained = 2
  Time used = 00:01:17 (CPU time = 00:01:06.4)

Tree scores written to file: ~/Desktop/PaupGit/documentation/supporting-documents/test.out

Frequency distribution of tree scores:

          mean=1369.568692 sd=66.080676 g1=-0.728020 g2=0.244030
1021.000  ---------------------------------------------------------------------------
1116.400 | (25477)
1211.800 |●●● (668259)
1307.200 |●●●●●●●●●●●●●●●●●●●●●●●● (5316550)
1402.600 |●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●● (15880624)
1498.000 |●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●● (12568515)
          ---------------------------------------------------------------------------
```

Figure 5.3 *Frequency distribution of tree lengths from Exhaustive search from Example 2.*

The shape of this distribution has been suggested to provide a measure of the strength of phylogenetic signal (Hillis, 1991; Huelsenbeck, 1991; Hillis and Huelsenbeck, 1992). In general, the more left-skewed the distribution (as quantified by increasing negativity of the g1 statistic), the greater is the amount of phylogenetic signal present in the data. However, as pointed out by Hillis and Huelsenbeck (1992) (see also Källersjö et al., 1992) the existence of signal says little about the nature of the signal; all of the "signal" may be confined to one or a few relatively uninteresting groups (e.g., mouse + rat in a study of tetrapod relationships).

5.4.2 Branch-and-bound algorithm

Fortunately, an exact algorithm for identifying all optimal trees that does not require exhaustive generation is available. The ***branch-and-bound*** method, frequently used to solve problems in combinatorial optimization, was apparently first applied to evolutionary trees by Hendy and Penny (1982). This method closely resembles the exhaustive search algorithm described above. In this procedure, a ***search tree*** is traversed in a "depth-first" sequence, as illustrated in the figure below. The root of the search tree (A) contains the only possible tree for the first three taxa. We first construct one of the three possible trees obtained by connecting taxon 4 to tree A, yielding tree B1. Then, to this tree, we connect taxon 5, yielding tree C1.1. (If the data set contained more than five terminal taxa, we would continue to join additional taxa in this manner until a tree containing all T taxa had been completed.) Now, we backtrack one node on the search tree (i.e., back to tree B1) and generate the second tree resulting from the addition of taxon 5 to tree B1 (= tree C1.2). When all five of the trees derivable from tree B1 (C1.1-C1.5) have been constructed, we backtrack all the way to tree A of the search tree and take the second path away from this node, leading to tree B2. As before, all five trees derivable from tree B2 (C2.1-C2.5) are constructed in turn. Then we backtrack once again to tree A and proceed down the third path toward trees C3.1-C3.5. Eventually, we will have constructed all of the possible trees culminating with tree C3.5. If the length of each tree containing all five taxa were evaluated at the time of its construction, we would have performed an exhaustive search equivalent to that described in the above section.

Suppose that L represents an upper bound on the length of the shortest tree(s). For the present, we can obtain L, for example, by evaluating a random tree; if we know that a tree of length L exists, then the length of the optimal tree(s) cannot exceed this value. If, as we are moving along a path of the search tree toward its tips (containing all T taxa), we encounter a tree whose length exceeds L, then we need proceed no

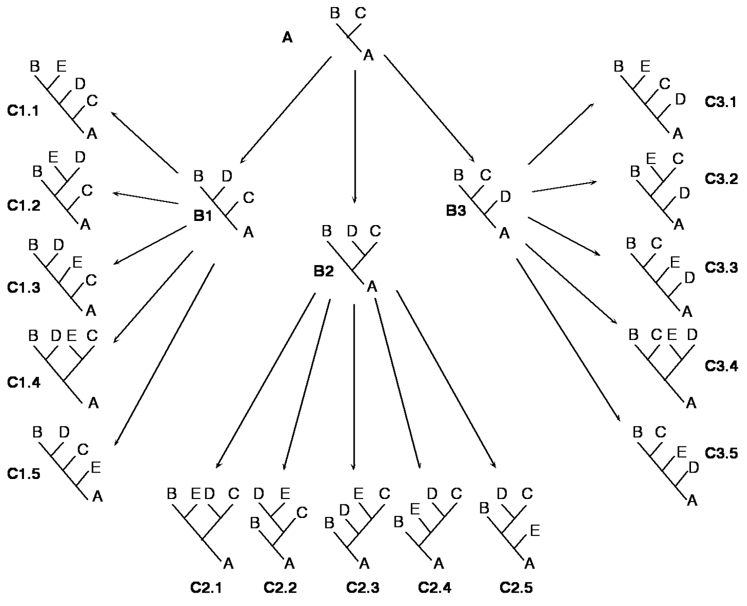


Figure 5.4 *Search tree for branch-and-bound algorithm.*

further along this path; connecting additional taxa cannot possibly decrease the length. Thus, we can dispense with the evaluation of all (phylogenetic) trees that descend from this node in the search tree and immediately backtrack and proceed down a different path. By cutting off portions of the search tree in this manner, we can greatly reduce the number of trees that must actually be evaluated.

If we reach the end of a path on the search tree and obtain a tree whose length is less than or equal to the upper bound L, then this tree is a candidate for optimality. If, however, this length is less than L, then this tree is the best one found so far, and we have improved the upper bound on the length of the optimal tree(s). This bound-improvement is important because it may enable other search paths to be terminated more quickly. When the entire search tree has been traversed, all optimal trees will have been identified.

Several factors influence the running time of the branch-and-bound algorithm. The quality of the data is perhaps the most important factor; large data sets with little homoplasy will run quickly because most paths of the search tree are terminated early. The speed with which the length of each tree can be evaluated, a function of the character types, is also important. In general, undirected character types run faster than directed types because certain algorithmic tricks for rapidly computing tree lengths can be used when the tree length does not depend on the location of the root. Ordered (Wagner) characters are much faster than unordered characters for similar reasons. User-defined stepmatrix characters, on the other hand, run very slowly due to the enormous amount of computation required to compute the length of even a single tree. Finally, of course, the speed of the available computer is critical to the run times-the branch-and-bound algorithm.

The above presentation of the branch-and-bound method, while correct, is an oversimplification of the algorithms actually used in PAUP*, which implements algorithmic refinements that greatly speed the computations. These refinements, designed to promote earlier cut-offs in the traversal of the search tree, include: (1) using heuristic methods (see below) to obtain a near-optimal tree whose length is used as the initial upper bound; (2) designing the search tree so that divergent taxa are added early, thereby increasing the length of the initial trees in the search path; and (3) using pairwise incompatibility to improve the lower bound on the length that will ultimately be required by trees descending from a tree at a given node of the search tree.

Since the branch-and-bound method requires evaluation of all trees as its worst possible case, why would we ever want to perform an exhaustive search? In fact, if we were interested only in the optimal trees, the branch-and-bound algorithm would indeed be the method of choice. However, exhaustive searches can be used to generate the frequency distribution of tree lengths. We may find it useful to know, for example, whether there are few or many near-optimal trees, or where some tree of prior interest lies in the distribution of tree lengths.

Example:

To begin a branch-and-bound search you must either specify the initial upper bound, for example the command

Branch-and-bound Search:

- Select **Analysis > Branch and Bound Search...**
- Set **Initial upper bound** to 1200 and click **Search**

This will set the upper bound at length 1200, or you can let PAUP* find it by stepwise addition.

Ordinarily, you will not need to specify an upper bound. However, the better the initial upper bound, the faster the branch-and-bound search will proceed. For large and/or messy data sets, you can sometimes reduce run times by performing a more extensive heuristic search and using the resulting tree length as the starting upper bound. The options for stepwise addition are slightly different than those available in a heuristic search, and include three choices: furthest, asis, or simple. PAUP* can also output a frequency distribution of tree lengths using the FD option or the **Tree length frequency distribution** item in the **Branch and Bound** dialog box. The maximum tree length to be shown must be specified by the Keep length. This option provides a way to examine the left tail of the distribution of tree lengths if there are too many taxa to obtain the full distribution via an exhaustive search (see below). To accomplish this, you can perform an initial branch-and-bound search to determine the minimum tree length and then perform a second search specifying FD and a Keep length some number of steps larger than the minimum length. In this way, you can determine the number of trees that are one step longer than the shortest, two steps longer, and so on. The output for a branch-and-bound search is nearly identical to that for a heuristic search except for when the FD is selected.

5.5 Heuristic Strategies

When a data set is too large to permit the use of exact methods, we must resort to heuristic approaches that sacrifice the guarantee of optimality in favor of reduced computing time. I like to apply the following analogy to the problem of searching for an optimal tree by approximate methods. Consider the plight of a myopic pilot who loses his glasses when forced to parachute from his airplane into a mountainous region. He suspects that there is a manned outpost at the top of the highest peak in the area, but he must somehow grope his way there to have any hope of rescue. Obviously, simply walking uphill from the point of landing will not necessarily lead him to his goal, since he may not have started on a slope of the highest peak. Suppose that he reaches a summit and finds no outpost. Two possibilities remain: (1) he is, in fact, at the top of the highest peak but was wrong about the existence of the outpost; or (2) he has climbed the wrong hill. Unfortunately, he will have no way of choosing between these alternatives. Although rather silly, the analogy is actually quite apropos.

Heuristic tree searches generally operate by hill-climbing methods. An initial tree is used to start the process; we then seek to improve the tree by rearranging it in a way that reduces its length. When we can find no way to further improve the tree, we stop. Like the downed pilot, however, we generally have no way of knowing whether we ended up at the top of the highest hill. That is, we do not know whether we have arrived at a **global optimum** or merely a **local optimum**.

Fortunately, the heuristic methods used in PAUP* are, so far as we can tell, very effective. Two basic strategies are used. An initial tree (or set of trees) is obtained by **stepwise addition**. Then the tree is subjected to trial rearrangements that attempt to find shorter trees. This second process is called, somewhat loosely, **branch swapping**.

5.5.1 Stepwise Addition

Stepwise addition operates by connecting taxa, one at a time, to a developing tree until all taxa have been placed. First, three taxa are chosen for the initial tree. Next, one of the remaining unplaced taxa is selected for next addition. Each of the three trees that would result from joining the unplaced taxon to the tree along one of its (three) branches is evaluated, and the one (in this example, but more can be saved at each step with the Hold option) whose length is optimal is saved for the next round. In this next round, yet another unplaced taxon is connected to the tree, this time to one of the five possible branches on the tree saved from the previous round. Again, one of the resulting trees is saved for the next round. The process terminates when all taxa have been joined to the tree. This process is illustrated in the figure below, where five taxa are added to a tree. First, three taxa are joined (A, B, and C). Next, taxon D is attached at each of three possible places. The shortest tree resulting from the addition of taxon D is retained, and taxon E is added, resulting in five trees. Branch swapping can then begin on the shortest tree(s) from these five.

Of course, the above description is oversimplified. We must have some way of determining which three taxa will be joined initially and which one of the unplaced taxa will be connected to the tree at each step. PAUP* provides four options for specifying the ***addition sequence***:

- *As is*. The taxa are simply added in the same order in which they are presented in the data matrix, starting with the first three and sequentially adding the rest. This method is usually not very effective.
- *Closest*. Initially, the lengths of all possible three-taxon trees-formed by joining a triplet of terminal taxa to a single internal node-are evaluated. The three taxa yielding compose the starting tree. At each successive step, all remaining unplaced taxa are considered for

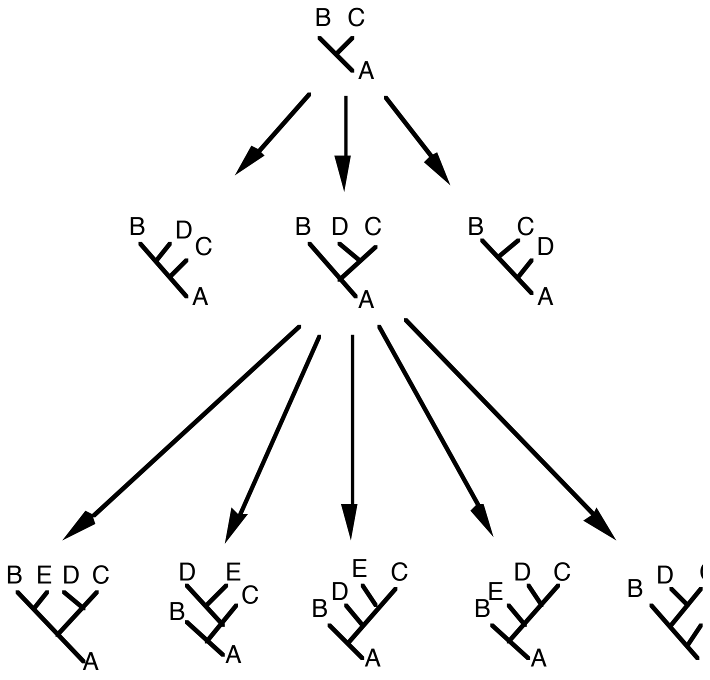


Figure 5.5 *Stepwise addition of five taxa.*

connection to every branch of the tree, and the taxon-branch combination that requires the smallest increase in tree length is chosen. Obviously, the closest approach requires considerably more computation than does *as is*. In the latter, the number of tree lengths that must be evaluated at each stage is simply equal to the number of branches on the tree (i.e., one for each potential connection point for the already chosen taxon). In the closest addition sequence, however, every unplaced taxon must be connected to the tree at every possible branch. In addition, each of the C(T, 3) possible triplets of taxa must be evaluated at the start.

- *Simple*. This option corresponds to the order in which taxa are connected in the "simple algorithm" of Farris (1970). As for as is, an addition sequence is determined prior to beginning the stepwise addition process; however, a more elaborate criterion for determining the sequence is used. First, the distance between each taxon and a reference taxon is calculated; Farris called this distance an ***advancement index***. The taxa are then added in order of increasing "advancement." That is, the reference taxon and the two taxa closest to it form the initial three-taxon tree, and the remaining taxa are added in the order given by their rank in the array of advancement indices. In Farris (1970), the reference taxon represented a "hypothetical ancestor" possessing the assumed ancestral state for each character. However, the algorithm can be used with any taxon chosen as the reference.
- *Random*. A pseudorandom number generator is used to obtain a permutation of the taxa to be used as the addition sequence.

When ties occur under the simple and closest addition sequences, they are broken arbitrarily.

Unfortunately, no one strategy seems to work best for all data sets; the best approach is to try as many alternatives as possible, each of which may potentially provide a different starting point for branch swapping (see below). In particular, although random addition sequences are not very effective in terms of the stepwise addition process, they are exceedingly useful in obtaining different starting points for branch swapping. For some data sets, "families" or "islands" of trees exist (Hendy et al., 1988; Maddison, 1991). Trees from the same island are much more similar to each other than trees from different islands. Suppose we define an island as a set of trees such that for every tree in the set, there is at least one other tree in the set that is exactly one rearrangement away [Maddison (1991); see "Branch Swapping"' for details on what constitutes a "rearrangement"]. Generally, if we find any tree in the island, we can recursively find all of the trees in the island by a process of swapping on every new tree we find[*]. However, by definition, a tree from a different island cannot possibly be obtained. Fortunately, by initiating branch swapping repeatedly from different starting trees, we can increase the probability of "beaching" on more than one island (and hopefully landing on all of them). One way of obtaining different starting trees would be to start from randomly generated tree topologies; however, these trees are usually so far from the optimal trees that searches are slow and ineffective. A compromise is to use trees obtained from random addition sequences, which are often different enough to be from distinct islands but still not too far from optimal. Another strategy would be to effectively "lower the water leve" between islands and swap on nonminimal trees. In that way you don't have to try and "drain the ocean" and swap on random trees.

Even when there is only one "island," random addition sequences can be used to circumvent the problem of entrapment in local optima. If a nonminimal tree cannot be rearranged to produce a shorter (or equal-length) tree, the rearrangement process ordinarily terminates, even though the globally optimal trees have not been found. By trying several different starting trees, however, the chances improve that at least some of these starting trees will lead to globally rather than locally optimal trees.

Finally, random addition sequences can be used to evaluate the effectiveness of the heuristics. If you do 100 different replications under the random addition sequences and get the same 15 trees in every case, you can be reasonably confident that these trees are in fact all of the optimal trees. By examining the "running status report" (available as an option), you can gain some idea of whether additional replications are likely to be effective. If, for example, the first 10 replications produce 50 trees in three different islands, and the next 20 replications terminate due to finding one of those 50 trees, the chances are reasonably good that you have found all of the islands and all of the most parsimonious trees contained in those islands. If, on the other hand, you are still finding new islands after 100 replications, a high probability exists that more trees remain to be found.

[*]This statement is strictly true only when the "collapse zero-length branches" option is declined. "Rearrangements" are defined only in terms of binary trees (see "Branch Swapping"). Before a tree in which zero-length branches have been collapsed is input to the swapping procedure, polytomous trees are first converted to binary trees by arbitrarily resolving polytomies into dichotomies. Although precautions are taken in the algorithm to avoid consistently resolving the polytomies in exactly the same way, this arbitrary resolution process can nonetheless prevent some trees in an island of binary trees from being found Maddison (1991).

The biggest drawback of stepwise addition algorithms is that they are too "greedy." Like the nearsighted pilot who is unable to scan the horizon and instead must simply proceed up the nearest hill, these methods strive for optimality given the current situation rather than attempting to look more broadly into the future. Thus, one placement of a taxon may be best, given the taxa currently on the tree, but that placement may become suboptimal upon the addition of subsequent taxa. Once a decision has been made to connect a taxon to a certain point, however, we must usually accept the consequences of that decision for the remainder of the stepwise addition process, perhaps ending up in a local optimum as a result. PAUP* provides one option that attempts to deal with excessive greediness. If the number of trees held at each step (Hold) is set to n, the n shortest trees from each step are considered, in turn, during the next step. Note that n trees are retained even if some are longer than others. For example, we might find in a given step trees of lengths 25, 25, 25, 26, 27, 27, and 28. The 26-step tree may ultimately be the best choice for retention, but if we saved only one of the 25-step trees, then it would be discarded prematurely. If Hold were 4 or greater, however, the 26-step tree would be retained and would eventually "rise to the top." Setting Hold >1 is also useful for minimizing the effect of ties early in the stepwise addition process, since ties are, to some extent, "followed out" rather than being broken arbitrarily.

In order to determine the length of the tree that would result if a taxon were to connect to the tree at a particular point, PAUP* evaluates the minimal length of the full tree, given that placement. (Some algorithmic tricks are used to minimize redundant calculations, however.) Note that this approach differs considerably from that of Farris (1970). In his method, character-state assignments made during each step are retained for all subsequent steps. These state assignments are only locally optimal given the state assignments from the previous iteration. Thus, a taxon might be connected to a branch to which it would not connect if a full optimization of the tree were performed.

5.5.2 Branch swapping

Because of the excessive greediness and susceptibility to local-optima problems, stepwise-addition algorithms generally do not find optimal trees unless the data are very clean. However, it may be possible to improve the initial estimate by performing sets of predefined rearrangements, a technique commonly referred to as "branch swapping." In general, any one of these rearrangements amounts to a "stab in the dark," but if a better tree exists and enough rearrangements are tried, one of them is likely to find it.

PAUP* uses three branch-swapping algorithms. In order of increasing effectiveness, these are (1) **nearest neighbor interchanges** (**NNI**, equivalent to the "local" procedure used in Versions 1 and 2 of PAUP); (2) **subtree pruning-regrafting** (**SPR**, approximately, but not exactly, equivalent to the "global" procedure used in earlier versions of PAUP); and (3) **tree bisection-reconnection** (**TBR**, a procedure added in version 3 of PAUP).

In NNI swapping, each internal branch of the tree defines a local region of four subtrees connected by the internal branch. Interchanging a subtree on one side of the branch with one from the other constitutes an NNI. Two such rearrangements are possible for each internal branch, as shown below:

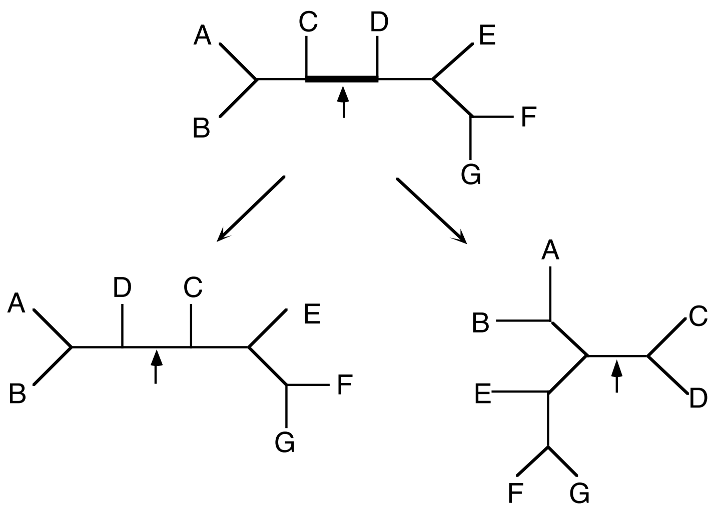


Figure 5.6 *Nearest neighbor interchanges around the branch partitioning taxa {A,B,C} from {D,E,F}.*

In "subtree pruning and regrafting," a subtree is pruned from the tree (e.g., the subtree containing terminal nodes A and B as indicated). The subtree is then regrafted to a different location on the tree. All possible subtree removals and reattachment points are evaluated.

In "tree bisection and reconnection," the tree is bisected along a branch, yielding two disjoint subtrees. The subtrees are then reconnected by joining a pair of branches, one from each subtree. All possible bisections

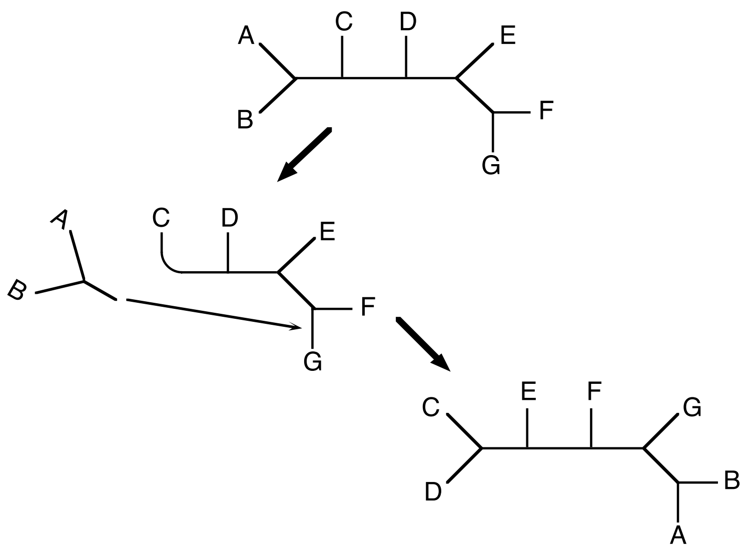


Figure 5.7 *An example rearrangement via "subtree pruning and regrafting." (a) the starting tree; (b) tree after pruning the subtree containing terminal taxa A and B; (c) tree after regrafting this subtree to the peripheral branch incident to terminal taxon G.*

and pairwise reconnections are evaluated.

Of course, the globally optimal tree(s) may be several rearrangements away from the starting tree. If a rearrangement is successful in finding a better tree, a round of rearrangements is initiated on this new tree. So long as each round of rearrangements successfully finds an improved tree (according to their length under the optimality criterion), then we will eventually arrive at the global optimum. However, if the path to the optimal trees requires us to pass through intermediate trees that are inferior to the best one(s) yet obtained, we will once again find ourselves trapped in a local optimum unless an option is provided for branch-swapping on suboptimal trees (the "Keep" option in PAUP* may be used for this purpose). A related problem concerns "plateaus" on the optimality surface. It may be the case, for example, that an optimal tree lies several rearrangements away from the current tree, and that these rearrangements all correspond to trees having equal lengths under the optimality criterion. If the intermediate trees are discarded because they are "not better," then the optimal tree will not be found. The MulPars option requests the saving of all of the equally most parsimonious trees.

Swapping on polytomous trees: The rearrangement algorithms above are defined only for dichotomous trees. However, when the Collapse (zero-length branches) option is in effect, trees recovered from memory for input to branch swapping may have one or more polytomous nodes. In this case, PAUP* "dichotomizes" the tree arbitrarily in order to prepare the tree for branch-swapping. This arbitrary resolution can introduce additional complications. Most importantly, the resolution of the polytomy that PAUP* chose may not lead to all minimal trees in an island. It may be that another resolution would actually have done better, but was not chosen.

There are two solutions to this problem. First, you can save all of the trees found by branch-swapping and input them into a second round of swapping. This greatly increases the chances of finding all of the trees in an island. The second solution is not to use the Collapse option in the first place. With Collapse off, you can save all dichotomous trees and use the Collapse command at the end of branch swapping. The problem with this strategy is that there may be too many dichotomous trees, and huge amounts of memory and long search times may be required.

The effect of using Collapse has special significance when random addition-sequences are used to construct starting trees. In the normal case, if PAUP* finds a tree that it has found in a previous replicate, it assumes that branch swapping in the previous round has already found all trees in the island to which that tree belongs; therefore it abandons the

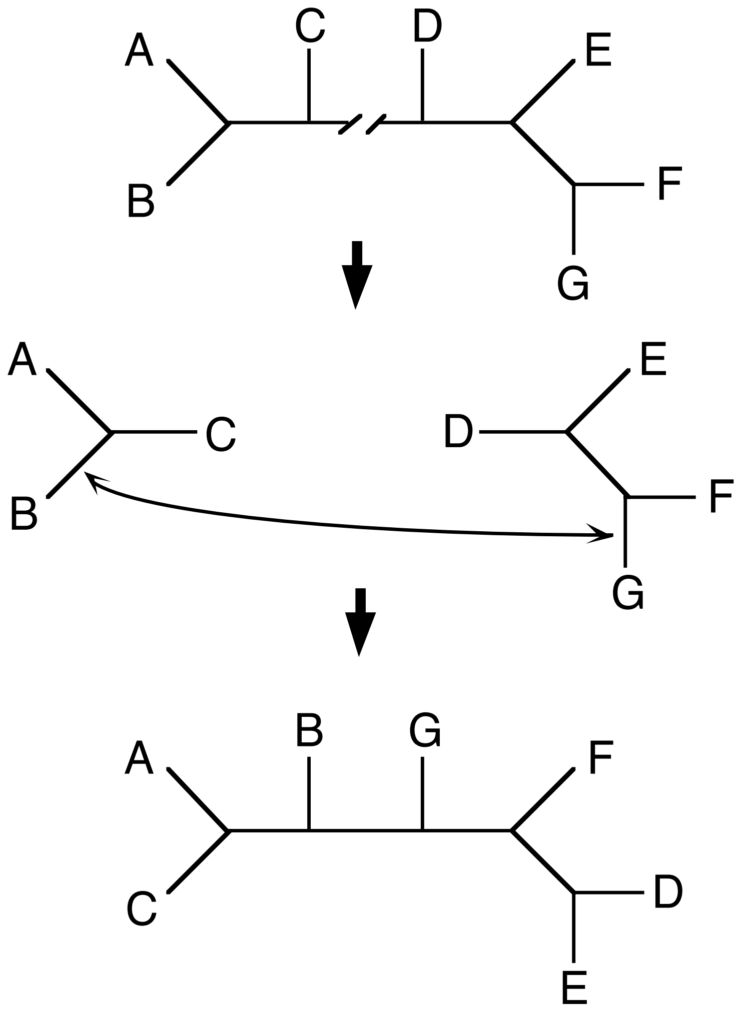


Figure 5.8 *Example rearrangement via tree bisection-reconnection. Initial tree is bisected into two subtrees and then reconnected along a different pair of branches.*

current round. But that assumption may not be true, since there is no guarantee that the original "dichotomization" allowed full exploration of the island. So the trees that resulted are not *necessarily* all of the minimal trees. In that case, it may be prudent not to use Collapse in conjunction with random replicates. This may greatly increase computation time, but will avoid the problem of arbitrarily missing parts of islands.

Steepest descent: The first "round" of swapping begins with the trees(s) in memory. These can come from several sources: user-defined trees, an external treefile, a Trees block in a data file, or from stepwise addition. In the simplest case, PAUP* will swap only on minimal trees (although this can be changed). In that case, the first round begins by discarding all nonminimal trees and swapping on the first of the minimal trees. For example, if we start with five trees in memory of length 30, 30, 30, 31, and 32, PAUP* will begin by discarding the two trees longer than 30 steps. It will then begin swapping on one of the 30-step trees. If no trees shorter than 30 steps are found, it will go on to swap on another 30-step tree. If Steepest is off, when a tree of less than 30 steps (say, 29 steps) is found *at any point*, all 30-step trees from that round are discarded. That 29-step tree is now the input for the next round of swapping. Subsequent rounds begin each time a shorter tree is found. The process terminates when swapping on all starting trees from a previous round does not find any shorter trees.

However, if the Steepest option is on, the round is not abandoned when a shorter tree is found, but continues until *all* trees from the previous round have been examined. In the example above, this means that the round would not end as soon as a 29-step tree was found. PAUP* would swap on all of the initial 30-step trees, saving all of the new shortest trees and passing them to the next round. It's called "steepest descent" because instead of discarding current trees the instant a shorter one is found, it continues looking for even shorter ones, and uses the tree(s) that give the most improvement for the next round.

Note that if Steepest is off and swapping on the first 30-step tree led to a 29-step tree, PAUP* would not swap on the other starting trees. Thus if you want to force swapping on all minimal starting trees, you must enable Steepest. Steepest descent is sometimes effective in finding other islands because it may allow the search to take a different path than it would otherwise. One way to hop islands is to swap on nonminimal as well as minimal trees; steepest descent is similar in that more nonminimal trees are tried as the search continues to find shorter and shorter trees. For many data sets, my experience is that steepest descent causes the program to get bogged down finding thousands of trees that will eventually be discarded anyway, and a better strategy is to perform more replicates of the random addition sequence without steepest descent.

5.5.3 *Performing a heuristic search*

Heuristic searches generally provide the fastest way to find optimal trees, but the results, being approximate, may depend on the way in which the search is conducted. Unlike the exact methods, heuristic methods may find only a subset of the optimal trees for a given data set and a given set of search parameters or may fail to identify minimum length trees at all. In addition, no one combination of settings will provide the best results for all data sets. You must be prepared to spend some time exploring different options to find optimal trees. This is especially true for large data sets, for which exact methods cannot be used. See the section "Heuristic Search Strategies" above.

To demonstrate several aspects of tree searching using heuristic methods, we are going to use another dataset. At this time execute the "cats.nex" data file of Masuda et al. (1996) included in the PAUP* distribution.

Execute cats dataset:

- Select **File > Open...**
- Select "cats.nex" data file and click **OK**

Alternatively, we could have just re-executed the data file by typing ⌘O to open a file.

First we will search for trees using the default heuristic settings. These settings will result in a search consisting of one round (rep) of "simple" stepwise addition and the TBR branch-swapping algorithm.

Example 1:

Default Heuristic Search:

- Select **Analysis > Heuristic Search ...**
- To make sure everything is set to the defaults, click on the **Default** button in the lower lefthand corner of the dialog box
- PAUP* will prompt you with another dialog box. Select the option to **Restore settings to current startup defaults** then click **OK**
- Now click **Search**.

PAUP* should have found 2 trees of equal length (336 steps). If you examine these trees (ShowTrees command), you will notice that they only differ in the position of the taxa labeled "European" and "Nafrican."

Example 2:

Next we will perform a heuristic search using multiple rounds of random stepwise addition.

Heuristic Search:

- Select **Analysis > Heuristic Search ...**
- Check **random** under the **Stepwise Addition** options and set **# reps** to 500 and click **Search**.

You will now notice in the display buffer that PAUP* has found multiple islands of trees (information given under "Tree-island profile:"). It shows here that PAUP* found 3 island, all of equal size (2 trees), but with two of them have trees with scores of 336, while the other has a less optimal score of 338. That is, there are two islands of equally parsimonious trees. We can examine the differences between the four trees using the ShowTrees command.

Examine Trees:

- Select **Trees > Show Trees ...**
- make sure that all four of the trees are highlighted and click **Show**.

You can also execute the ShowTrees command) by typing ⌘H. Like in the previous example, you will notice that 2 of the trees differ in the position of the taxa labeled "European" and "Nafrican." While the other two trees differ in regards to the position "Lion" and Cheetah."

There are too many options to give exhaustive examples of the HSearch command. At some point it is incumbent on the user to explore how changing options affects the analysis of her/his data set. The reason that PAUP* has all these features is to make it possible for you to explore the data set in as many ways as possible. Like any analytical program, PAUP* can only give you the tools - you must decide which tools to use and when to use them. Unfortunately, the bottom line is that no two data sets are the same, so no two heuristic searches will ever be the same. Searching involves changing the search options from the default values. By default zero-length branches are collapsed, all most-parsimonious trees are kept (MulPars), and steepest descent is not invoked. You can find out the current search settings at any time by typing the command:

```
hsearch ?;
```

In the command-line window at the bottom of the display buffer. This will list all options available during a heuristic search and their current settings. Options which are nonpersistent are marked with an asterisk (*).

Once you have selected the general options (listed above) that will be in effect during the search, you have other decisions to make: where to get starting trees for branch swapping; if by stepwise addition, how to add taxa; and how to branch swap. Starting trees for a search can be some or all trees in memory (using FromTree and ToTree options or by selecting tree numbers in the **Heuristics** dialog box), or they can be obtained by stepwise addition (using Stepwise or by selecting this in the **Heuristics** dialog box). Using trees in memory allows you to use the results of a previous analysis as the starting point for a new search. Any or all trees in memory may be used, so swapping can start on both optimal and suboptimal trees.

NOTE: *Selecting the starting tree range cannot be done if trees in memory have been filtered. You must either start with tree(s) obtained by stepwise addition or swap on all trees in memory*.

If trees in memory are not used, you must specify which addition sequence to use to obtain starting trees. You must also specify how many equal-length trees PAUP* will keep during each step of stepwise addition using the Hold option or by entering this value in the **Heuristics** dialog box. Stepwise addition is selected using the AddSeq option or by choosing it in the **Heuristics** dialog box. You may opt to add taxa in the sequence they are in (AsIs); to add the closest taxon at each step (Closest); by following the "simple algorithm" (Simple) of Farris (1970), or in random order (Random). The Simple option requires that you choose a "reference" taxon (default is the first taxon in the matrix-see the section "Stepwise Addition").

NOTE: *The reference taxon for simple addition sequence is ignored if the tree is rooted -in that case the ancestor serves that role*.

Random addition requires a few more options be set, specifically the number of replicates (NReps or the **#reps** selection in the search dialog box); a "seed" for the random number generator (by using the RSeed option or by entering this value in the **Heuristics** dialog box) and whether or not a running status of the random addition should be displayed (the RStatus option or by entering this value in the bf Heuristics dialog box). Hold values are ignored when random addition is used. The following command will begin a search with ten replicates of random addition and the initial seed set to 123:

Example 3:

Hueristic Search:

- Select **Analysis > Heuristic Search...**
- Choose **Stepwise-Addition Options** from the dialog box.
- Select **random** and set **# reps** to 20 and **seed** to 123
- Click **Search**

Once the starting trees have been obtained, the only thing remaining is selection of a branch swapping algorithm. These include **TBR** (tree bisection-reconnection), **NNI** (nearest-neighbor interchange), and **SPR** (subtree pruning-regrafting). These algorithms are described in detail in the section "Branch Swapping" The following command will begin a search with simple addition sequence and subtree-pruning-regrafting branch swapping:

Example 4:

Hueristic Search:

- Select **Analysis > Heuristic Search...**
- Choose **Stepwise-Addition Options** from the pull-down dialog.
- Check **simple**
- Choose **Swappong algorithm**.
- Check **SPR** under **Swapping options** and Click **Search**

5.6 Algorithmic Methods

In addition to criteria based methods, PAUP* provides several algorithm-based method to generate phylogenetic trees. Algorithmic methods generate trees by following a series of steps, unlike criterion-based methods define an optimality criterion (e.g., parsimony, maximum likelihood, and distance) for comparing alternative phylogenies to one another and deciding which hypothesis is preferred. The advantage of algorithmic methods are that they are computationally much faster than criteria-based methods due to the fact that they do not have to evaluate a large number of competing trees (see "Number of possible bifurcating trees").

5.6.1 *UPGMA*

Unweighted Pair Group Method with Arithmetic mean [UPGMA: Sokal and Michener (1958)] is a simple clustering algorithmic method for building phylogenetic trees (sometimes called dendrograms) that makes the often strong assumption that the data are ultrametric (i.e. completely clock-like). From a given distance matrix (see "Dse" command for more details on calculating a matrix), the two most similar units (or clusters) are combined into a new unit, or composite cluster. Next, from a new group of clusters (composite and simple), the pair with the highest similarity is identified and clustered. This process continues until two groups remain. UPGMA is an *ultrametric* method: that is, the path length from the root to any tip, is the same (i.e., fits the assumption of a 'molecular clock') and produces trees that are rooted (see "Outgroups, Ancestors, and Roots" for more detail) . The method assumes that the two most closely related clusters are more similar to each other than they are to any other. If this is not the case, spurious results may occur.

Violation of the assumption of a perfect-clock can cause UPGMA to be inconsistent. Indeed, its susceptibility to inconsistency due to violation of this principle, seems to be much larger than that of other methods which are also susceptible to unequal evolutionary rates. With short internal edges, it can be very difficult to detect where UPGMA may be going wrong, especially if the data are near ultrametric. Additionally, because of the methods susceptibility to systematic errors due to unequal rates, this methods bootstrap support for the branching order can also quickly become very distorted (so that an edge which is incorrect due to systematic error caused by unequal rates can quickly show 100% bootstrap support!).

UPGMA:

- Select **Analysis > Neighbor Joining/UPGMA...**
- Then select **UPGMA** under **Algorithm** and click **OK**

5.6.2 *Neighbor-joining*

Like UPGMA, neighbor-joining (Saitou and Nei, 1987) is another algorithmic method that uses clustering to build trees. Unlike the UPGMA method, neoghbor-joining does not require the data to be ultrametric. The method can also be used as an option for a starting tree

(Start = NJ) for heuristic searches (see the “HSearch” command for more detail).

Neighbor Joining (NJ):

- Select **Analysis > Neighbor Joining/UPGMA...**
- Then select **Neighbor joining** under **Algorithm** and click **OK**

PAUP* can also build trees using the BioNJ algorithm (Gascuel, 1997).

5.7 Quartet puzzling

Quartet puzzling (Strimmer and von Haeseler, 1996) is another method that is used to infer phylogenies available in PAUP*. Quarter puzzling goes about inferring a phylogeny for a set of N sequences by first calculating the maximum likelihood tree for all $\binom{N}{4}$ possible quartets (4 taxon trees). Then, during the *puzzling phase*, the quartets are made into a *intermediate* tree adding sequences one by one. Since the results this step may be biased do to the order in which the sequences are added, many different *intermediate* trees are made from multiple (n) sequence input orders. Finally, a majority rule consensus tree is made from the (n) trees from the second step.

Quartet Puzzling:

- Select **Analysis > Quartet Puzzling...**

5.8 Searching under Topological Constraints

Topological constraints are of two types in PAUP*: “monophyly” constraint trees, which contain all of the taxa in the data matrix, and “backbone” trees, which contain a subset of taxa. To search under constraints, you must define and select a constraint tree; this tree is used to restrict the set of trees retained by heuristic or exact searches.

5.8.1 "Monophyly" constraint trees

Monophyly constraint trees are usually incompletely resolved (i.e., contain one or more polytomies). These polytomies on the constraint tree indicate uncertainty with respect to relationships rather than simultaneous divergence into more than two descendant lineages. A tree being evaluated (the "trial tree") is said to be compatible with the constraint tree if and only if it is either identical to the constraint tree or it can be transformed into the constraint tree by deleting (collapsing) one or more of its branches. In more biological terms, constraint-tree compatibility means that any statement of relationship among taxa implied by the constraint tree must also be true for the trial tree. Thus, trial trees compatible with the constraint tree are (usually) "more highly resolved versions" or "refinements" of the constraint tree. For example, in the figure below, trial trees *b* and *c* satisfy the constraints imposed by tree *a* but trial trees *d* and *e* do not.

If a group of particular interest is not monophyletic on the minimal trees for your data set, constraining a search so that only those trees consistent with the group's monophyly are retained makes it easy to determine how much longer are the shortest trees on which the group *is* monophyletic. For example, if the most parsimonious tree for your data set does not support the monophyly of a genus according to someone else's classification (established on the basis of other data), you might be less inclined to challenge that classification if the genus were monophyletic on a tree only one step longer than if 28 additional steps were required for the group's monophyly. This notion can be extended to testing your data against an entire classification rather than only a single group. Since established classifications are usually not fully resolved, you can constrain the search so that only trees that are consistent with that classification are retained. Thus, where the classification is unambiguous, the relationships it implies must be maintained before a trial tree is accepted. However, other aspects of the classification can be resolved in the way that is most parsimonious for the data at hand.

Before the topological constraints feature was implemented in PAUP*, users could force the monophyly of particular groups by including "dummy" synapomorphies in the data matrix and weighting them so heavily that any tree on which the group was not monophyletic was immediately rejected. This approach was inconvenient and sometimes tedious, and had the further drawback that the dummy characters needed to be deleted before tree lengths and consistency indices were interpretable. Furthermore, it is impossible to determine the number of steps required to "break up" a monophyletic group appearing on the

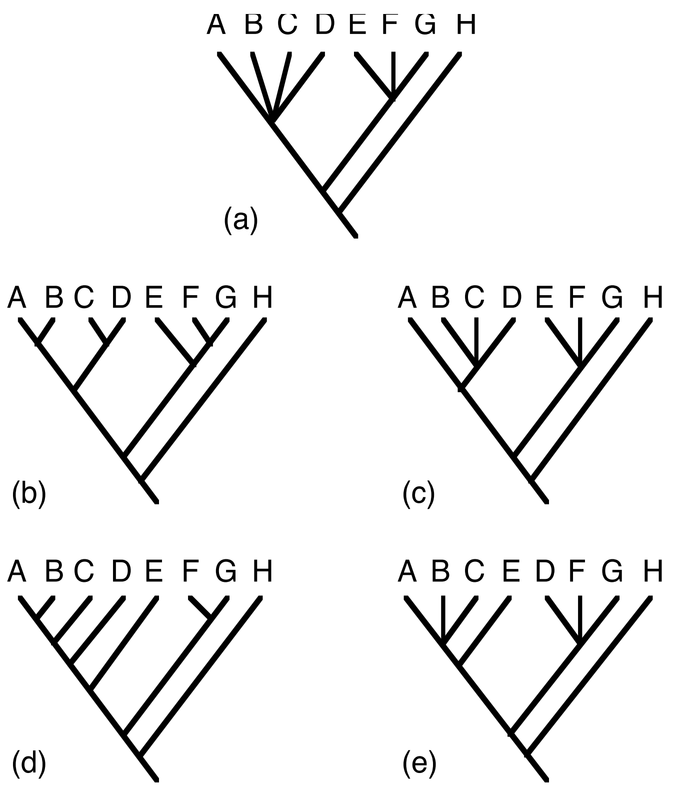


Figure 5.9 *Example for constraint-tree compatibility. (a) The constraint tree. (b,c) Two trees that are compatible with the constraint tree. (d,e) Two trees that are incompatible with the constraint tree.*

most parsimonious trees by using dummy character techniques. In addition to allowing you to force compatibility with the constraint tree, PAUP* allows you to request acceptance of trees only if they are incompatible with the constraint tree. This information provides a crude index to the strength of support for a clade. Of course, you are then faced with the decision as to how many steps longer a tree must be before it the existence of a clade is considered to be insupportable. (One way to make this judgment is through the use of the "T-PTP" randomization tests suggested by Faith (1991). **DLS: temper this recommendation**)

Note that although constraint trees are input as rooted trees, the criterion for satisfaction of topological constraints is the same whether rooted or unrooted trees are being determined. For unrooted trees, the rooted constraint tree is first "derooted." Then each tree being evaluated is compared to the (unrooted) constraint tree; if the trial tree is equal to the constraint tree or the trial tree can be converted to the constraint tree by deleting one or more branches from the trial tree, then the constraints are satisfied. An example is shown below.

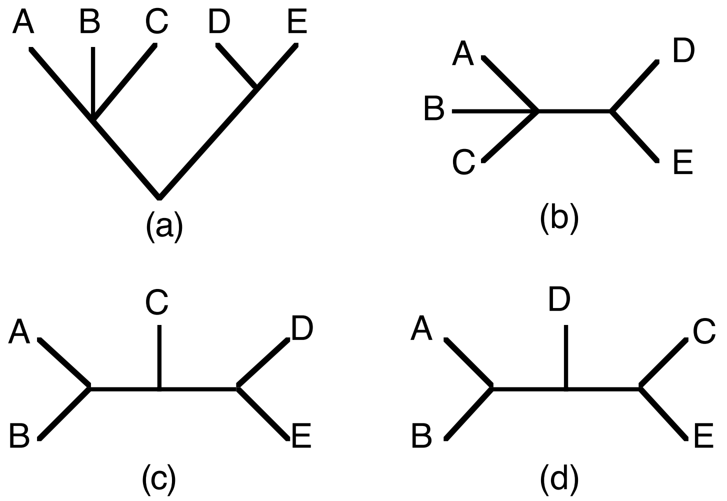


Figure 5.10 *Unrooted constraints. (a) A rooted constraint tree. (b) Unrooted equivalent of the constraint tree. (c) A tree that satisfies the constraints imposed by tree "b." (d) A tree that violates the constraints imposed by tree "b".*

Topological constraints can also be used to restrict the "solution space" when searching using exact algorithms (Sankoff et al., 1982;

Constantinescu and Sankoff, 1986). If the monophyly of particular groups is indisputable, the imposition of constraints that enforce these groupings can greatly reduce the number of possible trees that need to be examined, thereby extending the limits of usefulness of the exact algorithms. Of course, the "guarantee" of optimality is then conditional on the validity of the assumed prior groupings, but that sacrifice may be a relatively small price to pay in many situations. Yet another use of topological constraints is to enforce ingroup monophyly or partial outgroup structure. Recall that when potential outgroup taxa vary for characters that are informative with respect to ingroup relationships, one recourse is to include several outgroup taxa in the analysis. In this procedure, the ingroup node incident to the branch partitioning the ingroup from the outgroup becomes the root of the ingroup portion of the tree. If, however, the ingroup and outgroup taxa do not constitute a partition on the most parsimonious tree(s), you may wish to impose ingroup monophyly as a constraint. (Presumably, you would do this only if additional evidence supporting ingroup monophyly exists, but is not contained in the data set.) Simply define a constraint tree with a single group consisting of the ingroup taxa and enforce that constraint during a search. Another problem that sometimes arises when using this approach is the lack of sufficient information to adequately resolve outgroup relationships, with the result that a large number of equally parsimonious trees differing only in the relationships among outgroup taxa are found. Again, if there is support for some outgroup structure that comes from information not available in the data set, these aspects of the tree can be enforced through topological constraints. Finally, note that if you use the option to collapse zero-length branches (see "Zero-Length Branches and Polytomies") PAUP* will not collapse a branch if this action would result in violation of the constraints for a tree that would otherwise satisfy them.

5.8.2 "Backbone" constraint trees

"Backbone" constraint trees differ from "monophyly" trees in that they contain only a subset of the study taxa. A trial tree is compatible with the constraint tree if pruning the taxa not present on the backbone tree from the trial tree leaves a topology identical to the backbone. Unlike the monophyly constraint trees described above, backbone trees force a *relative* topology, and other taxa may be added at any point on the backbone tree, as long as the backbone is not violated. For example, a typical backbone topology might look like tree *a* below, and the corresponding unrooted backbone topology would look like tree *b*. Trees *c* and *d* are compatible with the backbone, while tree *e* is not-when taxa F, G, and H (i.e., those not present on the backbone tree) are pruned, Tree *f* results, which differs from Tree *b*.

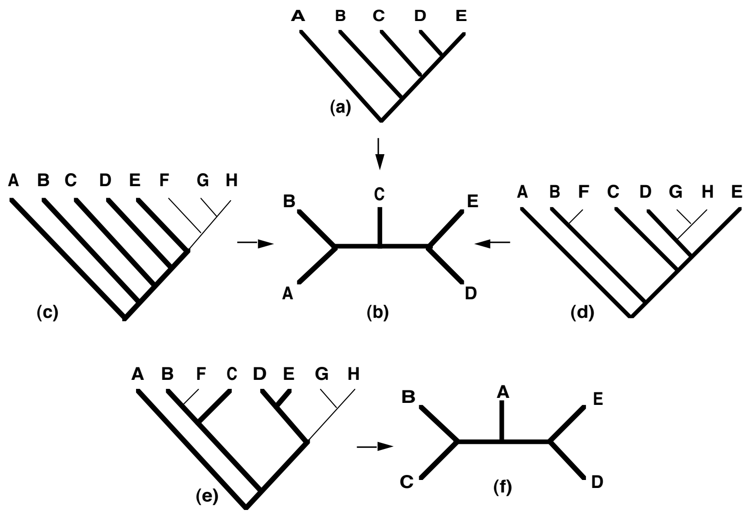


Figure 5.11 *Backbone constraints. (a) Rooted backbone. (b) unrooted equivalent of rooted backbone. (c) and (d) Rooted trees compatible with backbone-backbone is highlighted. (e) Rooted tree incompatible with backbone. Tree connecting taxa A-E is highlighted. (f) unrooted tree for taxa A-E derived from (e).*

Backbone constraints force a *relative* pattern of relationships. In the above example, the relative positions of A, B, C, D, and E on the tree must be the same. Using backbone constraints is less restrictive than using monophyletic constraints, in that any topology is compatible with the constraint tree as long as the relative backbone topology is preserved. This means that a backbone does not force an inclusive monophyletic group as does a monophyly constraint tree. The interesting feature of backbone trees is that they can also be unresolved (partially, as long as there are two child nodes-see above). The unresolved part of the backbone tree is even less restrictive, as even the relative topology within that part is not specified. In the end, whether you use a backbone or monophyletic constraint tree depends on your immediate goals-if you are interested in the lengths of trees which are compatible with a particular hypothesis of monophyly, use a monophyletic constraint tree; however if you are interested in which trees are compatible with a less restrictive topology, one that includes fewer taxa, use backbone constraints. In either case you may still be left with trees compatible with the constraint and those which are the shortest length, if they are different from the former group.

5.8.3 Heuristic searches and "converse" constraints

When "converse" constraints (i.e. trees are kept only if they are incompatible with the constraint tree) are used during a heuristic search, special problems may arise. This strategy will cause trouble if a heuristic search does not find the shortest trees that do not sashays the constraint, or if a heuristic search finds some, but not all, of those shortest trees. The latter is probably more common, but the former is more serious. If for whatever reason the data set is one that does not lend itself to a branch-and-bound analysis, it may be impossible to check whether either of these events has occurred, so some degree of skepticism might be warranted.

Why might these situations occur in the first place? The heuristic search uses the specified addition sequence, finding the shortest tree *until the last step*, when it *must* place the last taxon such that the constraints are violated. This will then be the starting point for branch-swapping. Because you have asked the algorithm to deliberately narrow the solution space before branch swapping even begins, it is possible that all optimal "converse" trees will not be found by branch swapping. You have simply asked the algorithm to begin swapping on a tree from which it cannot reach the shortest trees that are incompatible with the constraint tree. In some instances, trying different swapping and addition options may find more trees, but this is not always true. Again, if your data set is small enough to allow branch-and-bound searching, this will not be a problem. What are the solutions if an exact search is not possible? In that case you must be as diligent as possible in trying different addition sequences (especially random addition) and different swapping algorithms.

5.9 Keeping "Near-Minimal" Trees

You can keep trees less than or equal to a certain length by setting the **Keep all trees ≤ length <*value* >** option in the search procedure dialog boxes or specifying Keep = length in the HSearch, BandB, and AllTrees commands.

Selecting "steepest descent" forces the examination of suboptimal trees even when a shorter one has been found. This examination of suboptimal trees can be increased by selecting **Keep all trees ≤ length <*value* >**. This may allow the hopping of islands, so that the global optimum is reached, but can also substantially increase computation time.

One of the values of setting the **Keep all trees ≤ length <*value* >** option is that it allows one to examine the support for particular groupings in trees that are slightly longer than the optimal tree(s). We might, for example, keep all trees 3 steps longer than the optimal trees. If a particular group is present in the optimal tree(s) and all the suboptimal trees which are kept, then we might place more confidence in the support for that grouping. If, however, a particular clade is found only in the shortest trees and becomes nonmonophyletic in trees one or two steps longer, we might have less confidence in the support for that group.

The relationship between clade monophyly and tree length can be quickly evaluated by the use of topological constraints (see “Searching under topological constraints” above), but setting the “keep” option allows evaluation of the whole topology, not just those aspects set in the constraint tree (although as an alternative, multiple constraint trees can be examined). This strategy becomes more interesting when the frequency distribution of all possible tree lengths is examined (produced as a by-product of an exhaustive search, if that search type is possible for the data set (see the section “Exhaustive searches” for a description of this option). With detailed information about the number and lengths of all trees, we can better evaluate the strength of evidence for a particular group. In the example above, we kept all trees 3 steps longer than the optimal tree(s). However, the frequency distribution might show that there are few trees 3 steps longer, but many trees 4 steps longer, which might also be profitable to examine. In this way, it can be used in concert with the “keep” option in exploring suboptimal trees. As in the case of evaluating the length of trees which do not meet constraints of monophyly, you are still faced with the decision as to how many steps longer a tree must be before the existence of a clade is considered to be insupportable. Despite this difficulty, it is satisfying to know how long a tree must be before a particular clade becomes nonmonophyletic.

5.10 Defining and Using Topological Constraints

User-defined trees may also be used to enforce constraints during a search, or as a constraint in filtering trees in memory. There are two types of constraint trees: those that include all taxa in the matrix “monophyly” constraints, and those that include only a subset of those taxa “backbone” constraints. Both of these types must be defined or loaded before they can be invoked in a search. Defining constraint trees is done by using the Constraints command. The tree-description format is the same as for any other user-defined tree. See the section "User-defined trees" for details of tree description. An equivalent procedure is to use the **Load Constraints** menu command. This is quite useful-any tree in that has been previously saved may potentially be loaded as a constraint tree. These may include trees produced under different search conditions using the present data set, or trees from other studies of the same taxa. MacClade's (Maddison and Maddison, 1992) or Mesquite's (Maddison and Maddison, 2012) tree manipulation interface is particularly useful in constructing and defining different constraint trees.

5.10.1 *Loading constraints*

For this exercise we will switch back to the primate-mtDNA.nex dataset. Go ahead and re-execute it. Here we well only be dealing with a subset of the taxa. After you execute the file, go ahead and delete the non-hominoides (apes). Since we have defined these other taxa in a taxon set in the assumptions block this is easily done by:

Delete Taxa:

- Select **Data > Delete/Restore taxa...**
- under the **TaxSets** pull-down, select **otherSpp** and click **Delete**
- then hit the **OK**

If we do a quick search on this reduced dataset (either Exact of Heuristic) we se that PAUP* finds 1 trees of length 365. In this tree, chips are more closely related to gorillas than they are to humans, as in the tree on the right in Figure 5.12. Now we can load a constraint tree provided with the PAUP* distribution (constraint-file.tre).

Load a Constraint Tree:

- **Analysis > Load Constraints...**

  and select the file **constraint-file.tre**

Looking at the file you will see that the constraint trees are defined as NEXUS trees in a trees block of a NEXUS file.

Constraint Tree file:

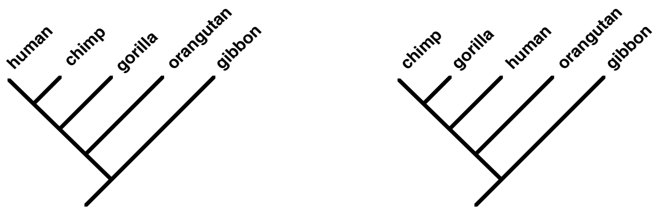


Figure 5.12 *Two alternative tree topologies.*

```
#NEXUS

begin trees;
    tree ch = ((human,chimp));
    tree chg = ((human,chimp,gorilla));
end;
```

Each of the trees define two mutually exclusive monophyletic group. The first "ch" defines a clade of chimp and human, then everything else, where the second constraint "chg" defines chimp, human, and gorilla to the exclusion of everything else. With a constraint this small, one might have also define these constraints by Typing in the command-line:

- `constraints ch = ((human,chimp));` and/ or
- `constraints chg = ((human,chimp, gorilla));`

In PAUP* the command (either on the command line or in PAUP block) for defining a "monophyly" constraint tree called "yourname" would look like

```
CONSTRAINTS yourname = tree-specification
```

where the tree-specification would follow the rules for user-defined trees above. Constraint trees are at least partially unresolved, otherwise only one tree would match the constraint. For example, the command

```
constraints hc=((human,chimp),gorilla,orangutan,gibbon);
```

defines an unresolved tree on which the only clade (other than the trivial all-species clade) is a group containing human + chimp.

As in the tree file you loaded, for monophyly constraints, you do not necessarily have to enter all taxa into the tree definition. This is because omitted taxa are joined to the root node of the subtree described by the tree specification for the included taxa. For example, to force the groups ((1,2),3), specify "constraints (((1,2),3))". Note that the outer pair of

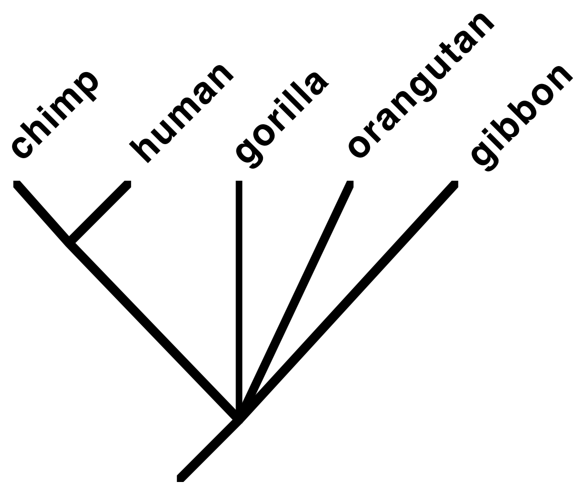


Figure 5.13 *A constraint tree.*

parentheses are necessary, otherwise the converted specification would be ((1,2),3,4,5,6) rather than (((1,2),3),4,5,6).

Constraints are enforced during a search by selecting this option in the search dialog box.

Searching with Monophyly Constraint

- Select **Analysis > Heuristic Search...**
- under the **General** settings, check the **Enforce topological constraints**
- Now select the **ch** constraint option from the pull-down
- Select the dialog so that PAUP* will **Keep trees that Are compatible with constraint** then hit the **Search**

Enforcing this constraint during the search will mean that any tree on which the human+chimp group does not appear is automatically rejected, regardless of its length. For the monophyly constraint above, the tree on the left below would be accepted, while the tree on the right would not.

You may also choose to retain only those trees which are *not* compatible with the constraint tree. In the first example above, this will retain all trees in which the human+chimp group does not appear. This is achieved by using the Converse and Enforce options during a search or by selecting this in the search dialog box.

Converse Constraint Search

- Select **Analysis > Heuristic Search...**
- under the **General** settings, check the **Enforce topological constraints**
- Now select the **ch** constraint option from the pull-down
- Select the dialog so that PAUP* will **Keep trees that Are Not compatible with constraint** then hit the **Search**

will keep trees not compatible with the constraint tree "test1."

Either type of constraint tree may also be used to filter trees after a search (though this is not the same as constraining during a search). In that case, the constraint trees are defined in the same way, and are invoked either by using the Filter command, or by the **Filter Trees** menu command; see the section "Filtering Trees." Complications can arise when using converse constraints with a heuristic search. See the section "Heuristic Searches and 'negative' constraints."

The rules for defining and enforcing backbone constraints are exactly the same as for monophyly constraint trees, with the exception that backbone trees always omit some of the taxa. When a backbone is specified, omitted taxa are left off the subtree. For example, the following command will define the constraint tree "bb1" as a backbone constraint:

```
  constraints bb1
backbone=((taxonA,taxonB),(taxonC,taxonD));
```

which is the equivalent of the following rooted constraint tree:

The tree below on the left is one of the rooted trees that satisfies the backbone constraint; the tree on the right does not satisfy it. See the section "Searching under topological constraints" for a full description of "monophyly" and "backbone" constraint trees, as well as the difference between rooted and unrooted constraints.

Multiple constraint trees of either type may be defined by separate Constraints commands, although only one of them may be the current constraint tree. This is conveniently done in either the data file or a command file, so that many different constraints can easily be tested on a given data matrix.

One benefit of using a backbone constraint is to limit tree search space to trees you may feel confident in as defined by the constraint, thus save

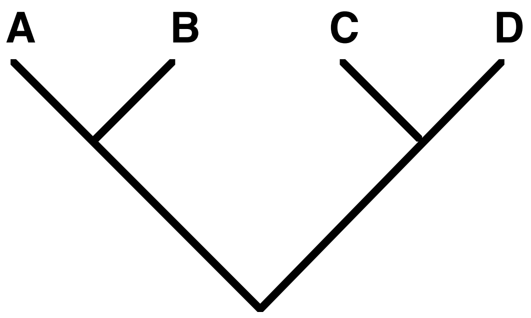


Figure 5.14 *Backbone constraint.*

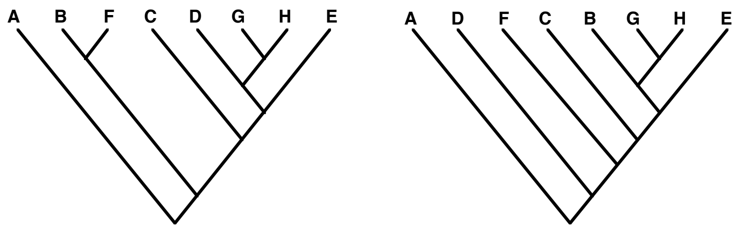


Figure 5.15 *Alternative constraint trees.*

time PAUP* spends evaluating suboptimal groupings. For our primate example, we could create a backbone constraint like the following:

```
  constraint bb backbone = (((human,
orangutan),Macaca_fuscata),Lemur_catta);
```

remember, this is the same as

```
 constraint bb backbone = (((2,5),7),1);
```

where the numbers correspond to the order the order the taxa are defined in the matrix. Now we can search under this backbone constraint.

Searching with Backbone Constraint

- Select **Analysis > Heuristic Search...**
- under the **General** settings, check the **Enforce topological constraints**
- Now select the **bb** constraint option from the pull-down
- Select the dialog so that PAUP* will **Keep trees that Are compatible with constraint** then hit the **Search**

When the search is completed, you will notice that PAUP* spent roughly one-third less time evaluating tree space.

5.11 Displaying constraints

Currently defined constraint trees are displayed using the ShowConstr command or the **Show Constraints** menu command.

Searching with Backbone Constraint

- Select **Analysis > Show Constraints...**
- Click **All**, the **OK**

PAUP* will then print all constraint trees in the display buffer. When this is invoked, "backbone" trees are flagged, and the number of trees compatible with all monophyly constraints are displayed (this is not displayed for backbone constraint trees due to the difficulty of evaluating it).

5.12 Successive weighting

In addition to *a priori* weighting, PAUP* allows *a posteriori* weighting based on the fit of the characters to the trees currently in memory. These *a posteriori* weights are then used as input for another (successive) analysis. You can continue to reweight and reanalyze until the weights do not change for two consecutive analyses or until identical trees (or sets of trees) are found in two consecutive searches. By doing this, you may or may not converge on fewer tree topologies, depending on the data matrix. If you choose this route, however, tree lengths will no longer be comparable between successive searches as they are directly dependent on the weights which are in place. The basic idea is to penalize characters which fit the tree(s) poorly (are homoplastic) and reward characters which fit the tree well. By successively evaluating and reweighting, the goal is to arrive at some stable topology or topologies. In practice, this amounts to running a search, choosing a weighting function, reweighting, and running another search. Reweighting can be based on the **consistency index** (**CI**), **retention index** (**RI**), or **rescaled consistency index** (**RC**). Since multiple most-parsimonious trees will generate different weight sets, PAUP* has the ability to weight by the best, worst, or mean fit to multiple trees.

It is up to the user to justify this approach, especially given the range of options under which it may be run. For example, the primary variable is the index chosen as the basis for reweighting, but it is not immediately clear which index should be preferred. You must also choose whether to reweight based on best, worst, or mean fit to the trees. (The default settings of best fit according to rescaled consistency correspond to the "xsteps w" command of the Hennig86 program (Farris, 1988)). Since different index and fit choices can potentially lead to very different trees, you must be very careful about conclusions based on a search strategy with many arbitrary components built in. If you do pursue this method, it is probably a good idea to experiment with different reweighting schemes, in order to get some crude idea of how stable the resulting topologies are.

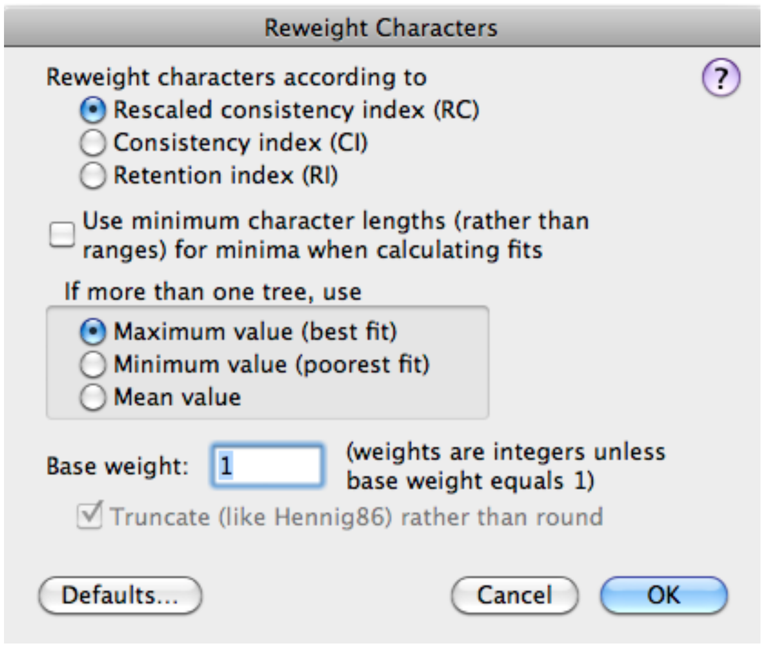
5.12.1 *Reweight*

Use the Reweight command to assign weights to the characters based on their fit to the trees currently in memory.

Currently defined constraint trees are displayed using the ShowConstr command or the **Show Constraints** menu command.

Reweight Characters

- Select **Data > Reweight Characters...**



Reweight characters according to

(● Rescaled consistency index (RC)) ○ Consistency index (CI) ○ Retention index (RI)

☐ Use minimum character lengths (rather than ranges) for minima when calculating fits

If more than one tree, use

(● Maximum value (best fit)) ○ Minimum value (poorest fit) ○ Mean value

Base weight: 1 (weights are integers unless base weight equals 1)

☑ Truncate (like Hennig86) rather than round

Figure 5.16 *Reweighting Characters Options Window.*

The simple command

```
reweight;
```

corresponds to the successive weighting method used in Hennig86 (Farris, 1988). The weight assigned to each character is proportional to the maximum rescaled consistency index over all trees in memory. The weights are scaled between 0 and the "base weight," which is initially set to 1000. The BaseWt option allows you to specify a different scaling. For example, if you issue the command

```
reweight basewt=10;
```

weights will be scaled between 0 and 10.

You can request that other fit measures be used via the Index option. Possible values for Index are RC (rescaled consistency index, the default), CI (consistency index), and RI (retention index). You can also use the Fit option to request that the minimum (worst fit) or mean fit values be used rather than the maximum (best fit). Available choices for Fit are Maximum, Minimum, and Mean. For example, to request reweighting using the mean (over trees) consistency index, you would use the command

```
reweight fit=maximum index=ci;
```

Ordinarily, after reweighting the characters, you will conduct another tree search using the new weights. The process of reweighting and searching continues until the weights no longer change (Farris, 1988) or until identical trees (or sets of trees) are found in two consecutive searches. You can use the `CSTATUS` command to examine the weights assigned to each character following a reweight command. This would be doen by typing:

```
cstatus full=yes;
```

or

Showing Character Status

- Select **Data > Show Characters Status...**
- Select the **Full details (one line per character)** option and clicking **OK**

5.13 Ancestral character state reconstruction

PAUP* provides two methods for reconstructing ancestral characters under the likelihood criterion - marginal and joint. The marginal method selects the character state assignment for a given interior node that has the highest posterior probability and corresponds to Equation # 4 in Yang et al. (1995). The joint method selects the complete set of ancestral character states with the highest posterior probability and corresponds to Equation # 2 in Yang et al. (1995). PAUP* uses an exact algorithm to reconstruct joint ancestral character sets similar to the one described by Popko et al. (2000). For the most part, marginal and joint methods will reconstruct the same set of ancestral sequences. However, when the probability of competing ancestral characters state assignments are close, it is possible to get different reconstructions under the two methods. To illustrate this point and to show how the two methods differ, consider the following example.

In this example we will reconstruct ancestral character states for the 4th position in the hypothetical nucleotide alignment given below.

```
4
t1 AGAAAAA ...
t2 AGAACAA ...
t3 GCAAAAC ...
t4 GCAACGA ...
```

The tree on which we will reconstruct characters is pictured below and the branch lengths are given adjacent to each branch. The two interior nodes are labeled a5 and a6.

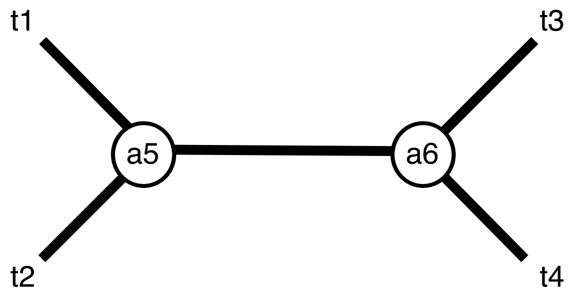


Figure 5.17 *Hypothetical 4 taxon tree with ancestral nodes labeled a5 and a6.*

Use the Reconstruct command to request output of character-state reconstructions for one or more characters on one or more trees. The reconstruction is shown by superimposing the character-states assigned to each node on a plot of the tree. Reconstructions are available under either the parsimony or likelihood criterion.

`reconstruct` [*character-list*] [/*options*];

The *character-list* specifies the character(s) for which reconstructions are shown, and consists of one or more character numbers, character names, or character-set names.

Showing Character Status

- Select **Trees > Show Reconstructions...**

Available Options:

- **Trees** = ***tree-list***

- **TCompress = Yes|No**

Description of options

Trees = *tree-list*

The *tree-list* specifies the tree numbers for which reconstructions are to be shown. If this is the first Reconstruct command and you do not specify a tree list, reconstructions are shown for the first tree only. If other options follow theTrees option you must the put the *tree-list* list in double quotes.

TCompress = Yes|No

See “Options And Subcommands Affecting Multiple Commands” for details.

If no characters are specified for either a Reconstruct or MPRsets command, the characters are taken to be those plotted in the last invocation of either of these commands. For example,

```
reconstruct 1 3 5 7; mprets;
```

will cause both commands to output information for characters 1, 3, 5, and 7.

5.14 Most Parsimonious Reconstruction sets (MPRSETS)

Use the MPRSets command to request output of possible character-state assignments (MPR-sets) for one or more characters on one or more trees, by superimposing the possible character-states for each node on a diagram of the tree.

Syntax: `MPRSets character-list [/options];`

The character-list specifies the character(s) for which possible character-state assignments are shown, and consists of one or more character numbers, character names, or character-set names.

Menu equivalent: **Trees > Show Reconstructions...**

Example:

- `Reconstruct 1 3 5 7;`
- `MPRSets;`

will cause both commands to output information for characters 1, 3, 5, and 7.

Available Options:

- **Trees** = ***tree-list***
- **TCompress = Yes|No**

Description of options

Trees = *tree-list*

The tree-list specifies the tree numbers for which possible character-state assignments are to be shown. If this is the first MPRSets command and you do not specify a tree list, reconstructions are shown for the first tree only.

TCompress = Yes|No

See "Options And Subcommands Affecting Multiple Commands" for details.

If no characters are specified for either a Reconstruct or MprSets command, the characters are taken to be those plotted in the last invocation of either of these commands.

5.14.1 Marginal Ancestral Reconstruction

To reconstruct ancestral character states using the marginal method, PAUP* compares the level of support for each characters state at each interior node and chooses the state assignment with the highest posterior probability. In practice, to obtain a measure of support for the assignment of an A at node a5 we must consider the possibility that node a6 might possess any of the four nucleotide states. More specifically, the likelihood of an A at node a5 is the sum of the likelihoods when an A, a C, a G, or a T is assigned to node a6 given that an A is assigned to node a5. The figure below illustrates this calculation.

aaaa

$7.12093e^{-4} + 7.82097e^{-6} + 8.56343e^{-7} + 8.56343e^{-7} = 7.21627e^{-4}$

The maximum likelihood estimate that an A is the ancestral state an the interior node a5 is $7.2167e^{-4}$. Next, the procedure is repeated for the remaining base assignments to the internal node a5. Because the tree used in this example only has two interior nodes, the partial likelihoods can conveniently be displayed in a 2 x 2 table, where the last column on the right of the table represents the marginal likelihoods for the interior node a5 and the bottom row represents the marginal likelihoods for the interior node a6. The posterior probabilities given below each likelihood score and are calculated by dividing the partial likelihood by the total site likelihood.

```
          NODE a6
NODEa5     A           C           G           T       Marginal for Node 5
    A 7.12093e-04 7.82097e-06 8.56343e-07 8.56343e-07 7.21627e-04
        0.48422     0.00532     0.00058     0.00058     0.49071
    C 1.86115e-05 7.12093e-04 1.32102e-06 1.32102e-06 7.33347e-04
        0.01266     0.48423     0.00090     0.00090     0.49868
    G 1.32102e-06 8.56343e-07 5.53415e-06 9.37639e-08 7.80528e-06
        0.00180     0.00058     0.00376     0.00006     0.00531
    T 1.32102e-06 8.56343e-07 9.37639e-08 5.53415e-06 7.80528e-06
        0.00180     0.00058     0.00006     0.00376     0.00531

Marginal for Node 6
        7.33347e-04 7.21627e-04 7.80528e-06 7.80528e-06 1.47058e-3
        0.49868     0.49071     0.00531     0.00531     1.0000
```

In PAUP*, you can obtain a list of the marginal posterior probabilities by setting the “allprobs” option under the “lsets” command and then executing the “describetrees” command with the “xout” option set to “internal”. For example:

```
lset allprobs=yes;

describetrees 1/xout=internal;
```

Typically, a complete list of probabilities for each site and each interior node would be displayed, however, for this example, we have only display the probabilities for site 4.

```
Relative probabilities of each base assignment
  Site    Node           A        C        G        T
-----------------------------------------------------
 4           5     0.49071  0.49868  0.00531  0.00531
             6     0.49868  0.49071  0.00531  0.00531
```

According to the output, the assignment of state C at node a5 has the highest probability relatively to the other three possible characters. and sate A has the highest probability at the interior node a6. Taken together, the marginal reconstruction of ancestral states looks like this:

5.14.2 *Joint Ancestral Reconstruction*

As the name implies, the joint method for reconstructing ancestral character states compares the support for the complete set of ancestral character sets. In this example, there are 16 possible ancestral character sets. Each of this 16 state sets are given in the table above as the interior cell values. Two state sets, AA and CC, are tied for the highest joint posterior probability and therefore represent the preferred joint reconstruction. Notice that the joint posterior probability (0.01266) for the preferred marginal reconstruction (i.e., C(a5) A(a6) is considerably less than posterior probability (0.48422) for the two preferred joint reconstructions.

In PAUP*, to get the ancestral character reconstructions using either the marginal or the joint reconstruction methods you will first need to set the optimality criterion to likelihood. For example:

```
set criterion = likelihood;
```

The "recon" option under the "lset" command controls allows the user to toggle between the marginal and the joint method of reconstructing ancestral character state. By default the marginal ancestral reconstruction method is used by PAUP*. Therefore, if you have not changed the default settings, the reconstruct command will display the preferred marginal reconstruction of a selected character. For example:

```
reconstruct 4;
```

```
Estimated states for character 4 on tree 1 (marginal estimation):

########################################################################## A t1
|
+------------------------------------------------------------------------- C t2
C
|                                    /------------------------------------ A t3
#####################################A
                                     ##################################### C t4

 Relative probability of this (joint) reconstruction = 0.01266
```

To get the preferred joint reconstruction, use the "lset" command with the "recon" option and execute the "reconstruct" command once again. For example:

```
lset recon=joint;

reconstruct 4;
```

```
Estimated states for character 4 on tree 1 (joint estimation):

########################################################################## A t1
|
+------------------------------------------------------------------------- C t2
C
|                                   ###################################### A t3
\-----------------------------------C
                                    \------------------------------------- C t4


  Relative probability of this reconstruction = 0.48422
```

For both of the reconstructions given above, PAUP* will display the joint likelihoods and the joint relative probabilities of each ancestral character state set.

6 Searching for Large Trees

As more and more sequence data has been collected over the past several decades, datasets have drastically increase in size with respect to, both, number of taxa, as well as number of characters. Today it is not uncommon to see datasets consisting of thousands (McMahon and Sanderson, 2006; Smith et al., 2009; Edwards et al., 2010) if not tens of thousand of sampled taxa (Smith et al., 2009). Because the number of possible trees grows exponentially with the number of taxa in a data matrix, new algorithms and software have been developed to deal with such large datasets.

Let Dave explicitly cite other programs

6.1 *The Parsimony Ratchet*

The **Parsimony Ratchet** is a search method proposed by (Nixon, 1999) for dealing with large datasets. The ratchet is an iterative search strategy that begins by generating a starting tree and performs an initial tree search (branch swapping), it then randomly re-weights a subset of the characters in the data matrix. Additional searching is then done on the trees in memory (from the initial search) with the re-assigned character weights. Next, the character weights are restored to there original values followed by more branch swapping. The procedure then performs multiple iterations of re-weighted and originally weighted search on trees.

Should we mention anything about Rice's paper? **Treezilla**, etc...?

6.2 ChuckScores and NChuck

Another way to estimate optimal trees is to use the NChuck/ ChuckLen option pair (so named for historical reasons) in conjunction with random addition (but see the note about the interaction of NChuck and Keep in the section below). If you set ChuckLen to some low value, such as one step, and set NChuck to ten, you will save no more than ten trees for each random replicate before the next replicate begins. If the next replicate finds ten different trees as short as the first ten, they will be added to the ten found before, and so on. In that way, you can build a set of trees in memory, say one thousand, with ten trees coming from each of one hundred random searches. This is a better estimate of the universe of optimal trees than saving one thousand trees from a single search, which probably gives a very biased estimate of the diversity of optimal trees.

For some data sets, you may want to try a huge number of random replicates but avoid wasting too much time on replicates that start with very suboptimal trees. You can specify AbortRep along with ChuckLen and NChuck to abandon a replicate as soon as the "chucking" limits are hit. In this case, many replicates are likely to be abandoned early, but many more (perhaps thousands of) replicates can be performed.

EXAMPLE

set maxtrees = 1000 increase=no; hsearch addseq=random nreps=10 nchuck=100 chuckscore=1;

In addition to Keep, heuristic searches also have the option of limiting the number of trees (NChuck option) greater than or equal to a specified length (ChuckLen option) which will be kept (see below). This is different from Keep, which specifies that all trees less than or equal a certain length are kept.

Example 1:

In the following example of the HSearch command the NChuck and ChuckScore options are used to save a maximum of 100 trees with scores of 600 or above; no limit is imposed for trees less than 600.

Heuristic Search:

- Select **Analysis > Heuristic Search...**

- Select **Branch-Swapping Options** from the pull-down menu.
- Check box **Save no more than** Type: `100` **trees** ≥ **score** Type: `600`
- Select **Search**

will begin a heuristic search, keeping no more than twenty-five trees of fifty or more steps. If Keep and NChuck are used together, once PAUP* reaches the limit specified by NChuck and ChuckLen, it stops looking for shorter trees; that is, it doesn't replace longer trees satisfying the Keep limit with shorter trees satisfying the Keep limit. When that happens, PAUP* will either stop or go on to the next step, depending on the circumstances.

Example 2:

Heuristic Search:

- Select **Analysis > Heuristic Search...**
- Select **General Search Options** from the pull-down menu.
- Check box **Keep** Type: `40` **best trees**
- Select **Branch-Swapping Options** from the pull-down menu.
- Check box **Save no more than** Type: `25` **trees** ≥ **score** Type: `50`
- Select **Search**

will try to keep all trees less than forty steps, but if twenty-five trees of length greater than or equal to fifty steps are found first, PAUP* will stop.

Part of the difficulty in using heuristic searches is that there are no settings that will get the best results for all data sets. Each search will be more or less unique, although different types of search will be similar. The choice of options will almost always affect the outcome of the search. This becomes critical when the data set is one which produces a very large number of trees. In that instance, there are several strategies for trying to get the best estimate of the set of shortest trees.

First, a simple illustration of how option choice can affect the number of optimal trees found. This example uses the sample Menidia data set distributed with the program. If this data set is analyzed with the following command

Example 3:

Heuristic Search:

- Select **Analysis > Heuristic Search...**
- Select **General Search Options** from the pull-down menu.
- Check box **Keep** Type: `1` **best trees**
- Select **Stepwise-addition Options** from the pull-down menu and click **asis**
- Select **Branch-Swapping Options** from the pull-down menu and click **NNI**
- Select **Search**

ten trees are found. If we now choose to hold ten trees instead of 1, the search now finds 24 trees (note that the Addseq = AsIs and Swap = NNI settings are automatically retained.

Example 7:

Heuristic Search:

- Select **Analysis > Heuristic Search...**
- Select **General Search Options** from the pull-down menu.
- Check box **Keep** Type: `10` **best trees**
- Select **Search**

Finally, switching to TBR swapping yields 25 trees. This is the maximum number that any other combination of settings will find in a heuristic search, as you can verify yourself by exploring the data set with other settings.

Heuristic Search:

- Select **Analysis > Heuristic Search...**
- Select **Branch-Swapping Options** from the pull-down menu and click **TBR**
- Select **Search**

Of course, with other data sets you may reach a stage where trying further rearrangements does not yield any new trees and takes up vast amounts of computer time.

In general, the best way to identify all optimal trees is to aggressively vary the swapping and addition-sequence options to explore as much of tree space as possible. The single best strategy is use of the random addition-sequence option, which will present the rearrangement algorithms with a broad range of starting trees on which to swap. The tradeoff is that random trees can be very far from optimal, and as such are not very good places to start to find optimal trees. It may drastically increase computation time and resources to find minimal trees from a random beginning. On the other hand, random addition frequently leads to the finding of more (if not all) islands than other addition sequences. You can also hold more trees during stepwise addition, which may increase the effectiveness of the search but also greatly increases the time spent on adding taxa.

Another strategy is to invoke steepest descent, which will not abandon a round of swapping until all minimum trees from the previous round have been evaluated; it then chooses the best available tree(s) for the next round. (Without steepest descent, a round of swapping ends as soon as a shorter tree is found, and this tree is used as the starting tree for the next round). Because the rearrangements producing greater improvements are preferred over those leading to smaller ones, use of steepest descent can sometimes reduce the problem of entrapment in local optima. Unfortunately, being overly ″greedy″ can also produce the opposite result-in some cases, rearrangements with smaller decreases in tree length ultimately lead to the shortest trees, whereas the rearrangements preferred under steepest descent represent blind alleys (local optima).

Also mention steepest descent as a way of avoiding ″incomplete islands″ (e.g., the ROPA 62-63-64 example).

You can also potentially increase your chances of finding minimal trees by swapping on nonminimal trees. The first way to do this is to Keep trees above a certain length, say one or two steps above the shortest tree found so far (be aware that you may end up retaining enormous numbers of trees when you do this). Then you can begin swapping on those trees. This may have some of the same problems as using random-addition sequence, but may in fact ″lower the water level″ between islands enough to allow PAUP* to find more of the optimal trees.

7 Manipulating and Summarizing Trees

7.1 Saving Trees to Files

If the search required to find a given set of trees uses a significant amount of computing time, saving the descriptions of these trees to a ”tree file” may be desirable for several reasons. First, you may need the trees at a subsequent time (e.g., to obtain additional output information about the trees, to plot the trees in a different way, or to use the trees as input for a subsequent search) after you have terminated the PAUP* session. If you have saved the trees to a file, you can then recover the trees quickly without having to repeat the lengthy search. Second, the tree file can be input to other programs that read user-defined trees. Since the trees are written to the file as a NEXUS Trees block, any NEXUS-conforming program that accepts user-defined trees (e.g., MacClade) will be able to read the tree file. Other programs that follow the standard format for tree description described in the section Describing Trees (e.g., PHYLIP, PLOTGRAM) will be able to interpret the tree file after minor editing. (Note that PAUP*'s export facilities, described in the section ”Importing and Exporting Trees and Data” provide a more direct means of inputting trees found by PAUP* to PHYLIP and Hennig86). Third, data files and tree files are standard text files that are portable across computers. For large and/or complex data sets, you may want to perform the searches on a larger, more powerful, computer, and then download the trees to a microcomputer for further analysis, graphics output, etc.

Use the Savetrees command to write trees currently in memory to a file as a NEXUS-format Trees block or as a treefile accepted by another program.

Example 1:

Saving Trees to File:

- Select **Trees > Save Trees To File...**
- Type: `trees.out` in the **Save treefile as:** dialog box
- Select **More options...** button.
- Under the **Save** section of the dialog box, click the **Trees** option and Type: `1` **though** `10` in the available boxes and click **OK**
- Click the **Save** button.

This will save trees one through ten to the file "trees.out" along a translation table that maps positive integers to taxon names. In essence, this creates nothing more than a Trees block contained in a single file. Additionally, trees may be saved rooted (or unrooted) although this may slow down the saving process.

Example 2:

Saving Trees to File:

- Select **Trees > Save Trees To File...**
- Type: `trees.out` in the **Save treefile as:** dialog box
- Select **Options...** button.
- Under the **Save** section of the dialog box, click the **Trees** option and Type: `1` **though** `10` in the available boxes
- Finally, check the **Save as rooted trees** option and click **OK**
- Since the file "trees.out" already exists, you will be asked if you want to **Append**, **Replace**, the file or **Cancel**. Click **Replace**

will replace the contents of the file "trees.out" with rooted trees one through ten.

PAUP* also includes information about the analysis that generated the trees, should you need to replicate the search. Here is a sample treefile resulting from a branch-and-bound search:

A TREES Block

```
#NEXUS

begin trees;  [Treefile saved Tuesday, February 16, 1993
    10:18 AM]
[!> Branch-and-bound search settings:
>   Initial upper bound: unknown (compute via stepwise)
>   Addition sequence: furthest
>   Initial MAXTREES setting = 100
>   Branches having maximum length zero collapsed to yield
    polytomies
>   Topological constraints not enforced
>   Trees are unrooted
>   Shortest tree found = 23
>   Number of trees retained = 5
]
utree PAUP_1 = (taxonA,(taxonB,(taxonC,(((taxonD,(taxonG,
    taxonH)),taxonE),taxonF))));
utree PAUP_2 = (taxonA,(taxonB,(taxonC,((taxonD,(taxonG,
    taxonH)),(taxonE,taxonF)))));
utree PAUP_3 = (taxonA,(taxonB,(taxonC,(taxonD,(taxonE,(
    taxonF,(taxonG,taxonH)))))));
utree PAUP_4 = (taxonA,(taxonB,(taxonC,((taxonD,(taxonF,(
    taxonG,taxonH))),taxonE))));
utree PAUP_5 = (taxonA,(taxonB,(taxonC,((taxonD,taxonE),(
    taxonF,(taxonG,taxonH))))));
end;
```

PAUP* can save trees in a variety of formats, however it is up to the user to specify the type of tree-file to produce under the Format pulldown in the Save window. The default format setting is Nexus. Nexus requests a file containing the standard NEXUS Trees block using a translation table (which greatly reduces the amount of disk space required to store the trees). AltNex also specifies a NEXUS Trees block, but no translation table is used (the full taxon names are included in each tree description). FreqPars requests a treefile for the FreqPars program described by Swofford and Berlocher (1987). PHYLIP requests a treefile for input to version 3.4 of the PHYLIP package by Felsenstein (1991) . HENNIG requests a treefile for version 1.5 of the Hennig86 program by Farris (1988).

In addition to the specific tree format, there are a variety of formating options available in PAUP*. One useful command is the BrLens, or "Include branch lengths" option. Users can access this option by clicking the More Options... button in the SaveTrees window. If this command is specified, tree descriptions will include branch lengths if the program

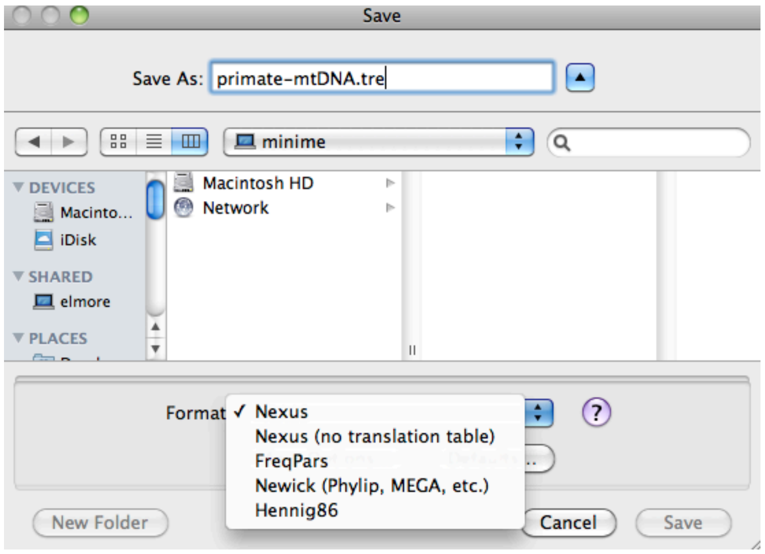


Figure 7.1 *Format options in PAUP* under the SaveTrees command.*

corresponding to the Format setting supports them. BrLens = NO reverses the effect of a previous BrLens specification. It is important to remember that branch lengths are calculated according to the current optimality criterion.

7.2 Recovering Trees from Files

Once trees have been saved to a file, PAUP* allows you to recover these trees back into the programs memory for further analyses. One or more of these imported trees may then be used as starting trees for searches, or the current data matrix may be optimized onto imported trees. PAUP* gives you the option of importing some or all of the trees in a particular treefile, or only those which meet certain conditions. This last category is available through Boolean operations.

Use the GetTrees command to load trees into memory from a file containing a NEXUS-format Trees block.

Example 1:

Recovering Trees from File:

- Select **Trees > Get Trees from File...**
- Select appropriate file and click **Options...**
- Different tree-number options and boolean operations are selected in the by manipulating two partially intersecting circles (ven diagram), one representing "Trees currently in memory", one representing "Trees from file" that you have selected to open.
- Choose the trees you want and click **OK**
- Click **Open**

Manipulating trees in this way gives you the ability to compare sets of trees, such as those derived from different data sets, different search options, or different assumptions about characters (e.g. weight). One you have imported and kept the trees you want, you can then save them to a different treefile. This allows you to have different sets of trees in different treefiles, and to extract different subsets of trees from each of those different treefiles.

PAUP* will echo the Boolean selection and display a summary of how many trees in memory and/or the file have been kept. For example, adding all trees in the file to the trees currently in memory gives this output:

Recovering Trees from File:

```
Processing TREES block from file 'manual.trees':
   Keeping: trees in memory plus trees in file (without
       duplication)
   6 trees originally in memory
   11 trees read from file
   5 trees from file kept
   11 trees now in memory
```

A GetTrees command specifies either the range of trees to be imported or the boolean Mode. Setting the Mode to different values invokes the different boolean operations described above.

IMPORTANT: *Getting trees using* Mode=2 *can be dangerous! If no trees in memory match trees in file, all trees in memory will be lost. You may want to save trees to a file before getting trees with this option.*

Example 2:

The following example of the GetTrees command reads the trees and branch length information (if present) from the file ”treefile.tre” without overwriting the trees currently in memory.

```
gettrees file=treefile.tre mode=7 storebrlens=yes;
```

Example 3:

```
gettrees file=trees.out
```

will import all of the trees in the file ”trees.out.” However you don’t have to import all of the trees in the file - you can also specify the tree numbers, if you know which number corresponds to which topology.

Example 4:

```
gettrees file=trees.out from=1 to=5;
```

will replace any trees in memory with trees one through five in the file ”trees.out.” These trees may be the products of a previous search of the current data matrix, they may be derived from a different data matrix for the same taxa, or they may have been created by a program such as MacClade. You can then test how certain starting trees affect the search, and explore how the characters are reconstructed on different trees.

NOTE: *You can also use the Import File command to get trees created by other programs, such as Hennig86 or PHYLIP. Once these have been imported into PAUP*, they can either be stored as NEXUS files, or exported to the original or a different format.*

By default (Mode=3), any preexisting trees in memory are replaced by the trees read from the file. The Mode setting allows you to alter this behavior. Let M = the set of trees originally in memory and T = the set of trees from the tree file. The following mode values are then available:

1 = replace M by T - M (i.e., keep trees from the file that are not originally in memory)

2 = replace M by T ¬¥ M (keep trees from the file that are also originally in memory)

3 = replace M by T (i.e., replace all trees in memory by all trees from the file)

4 = replace M by M - T (i.e., keep trees in memory that are not also in the file)

5 = replace M by M T (i.e., keep trees that are either currently in memory or in the file, but not both places)

7 = replace M by M ¬™ T (i.e., append trees from file to trees originally in memory)

7.3 Displaying and Printing Trees

TO DO: FIX THIS! There are two ways to output trees: plotting and printing. Plotting involves invoking the Plot option in a Describe command, or with the **Describe Trees** menu command. The output can be a cladogram, phylogram, or both. The destination of the plotted trees can either be a log file (when Log or **Log Output to Disk** is invoked) or a printer (when TO DO: SET? Echo or the **Echo to Printer** menu commandis invoked). The trees can be output in compressed or uncompressed form (when the TCompress option is chosen or by selecting it in the appropriate dialog box). Here is an example of an uncompressed and compressed tree after plotting. Compressing can save a great deal of space, especially when there is a large number of taxa.

```
describe 1;
```

would output an uncompressed tree:

```
?<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< A
>
>                    ?<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< B
>                    >
?<<<<<<<<<<<<<<<<<<<14      ?<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< C
                     >      >
                     ?<<<<<13         ?<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< D
                            >         >
                            ?<<<<<<<<12          ?<<<<<<<<<<<<<<<<<<< E
                                      >          >
                                      ?<<<<<<<<<11         ?<<<<<<<<< F
                                                 >         >
                                                 ?<<<<<<<<10    <<<<< G
                                                           ><<<<9<<<< H
```

while

```
describe 1/tcompress;
```

will output a compressed tree

```
?<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< A
>         ?<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< B
?<<<<<<<<14          ?<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< C
          ?<<<<<<<<<13         ?<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< D
                     ?<<<<<<<<12          ?<<<<<<<<<<<<<<<<<<<<<<<<< E
```

```
                               ?<<<<<<<<<11         ?<<<<<<<<<<<<<<< F
                                          ?<<<<<<<<10          ?<<<< G
                                                    ?<<<<<<<<<<9<<<< H
```

Trees output by plotting or printing cannot be recovered for further manipulation and analysis - for that they must be saved to a NEXUS file (see Saving Trees to Files above).

7.4 Changing the Order of Taxa on Trees

As is generally the case for phylogenetic trees, PAUP*'s trees are formally ”unordered” (see ”Tree Terms” in Chapter 1), however they may be ordered according to one of four available conventions for output purposes. You can specify which ordering to use by the TOrder option of the Set command or the Tree Order menu command. Because systematists tend to think of the root as the lowest point in the tree diagram, assume that the output has been rotated 90,àû clockwise, so that ”left” is actually up, ”right” is down, etc.

If TOrder = Standard, trees are ordered so that taxon names appear from left to right in, as nearly as possible, the same order as the taxa were presented in the data matrix. If TOrder = Right, the tree is ”ladderized” (a term used in MacClade) to the right. (Technically, this means that rotations are performed around each internal node so that the descendant node with the greatest number of ultimate descendants is placed on the right and the descendant node with the fewest descendants is placed on the left. TOrder = Left is just the opposite of TOrder = Right. Finally, if TOrder = Alphabet, the tree is ordered so that the taxon names appear, as nearly as possible, in alphabetical order, regardless of the order in which the taxa were presented in the data matrix. Examples of the same tree in different orders are:

TOrder = Standard

```
         ?<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< taxonA
         >         ?<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< taxonB
?<<<<<<<<?         >                           ?<<<<<<<<<<<<<<< taxonC
>        ?<<<<<<<<<?                  ?<<<<<<<<<<<<<<<<<<<<<<<< taxonF
>                  >        ?<<<<<<<<<?        ?<<<<<<<<<<<<<<< taxonG
>                  ?<<<<<<<<?         ?<<<<<<<<<<<<<<<<<<<<<<<< taxonD
>                           ?<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< taxonE
?<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< outgroup
```

TOrder = Right

```
?<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< outgroup
>        ?<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< taxonA
?<<<<<<<<?         ?<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< taxonB
         ?<<<<<<<<<?        ?<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< taxonE
                   ?<<<<<<<<?         ?<<<<<<<<<<<<<<<<<<<<<<<< taxonD
                            ?<<<<<<<<<?        ?<<<<<<<<<<<<<<< taxonC
                                      ?<<<<<<<<?         ?<<<<< taxonF
                                               ?<<<<<<<<<,<<<<< taxonG
```

TOrder = Left

```
                                                    <?<<<<<<<<< taxonF
                                               ?<<<<<<<<<,<<<<< taxonG
                                      ?<<<<<<<<,<<<<<<<<<<<<<<< taxonC
                            ?<<<<<<<<<,<<<<<<<<<<<<<<<<<<<<<<<< taxonD
                   ?<<<<<<<<,<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< taxonE
         ?<<<<<<<<<,<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< taxonB
?<<<<<<<<,<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< taxonA
?<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< outgroup
```

TOrder = Alphabet

```
?<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< outgroup
>        ?<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< taxonA
>        >         ?<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< taxonB
?<<<<<<<<?         >                           ?<<<<<<<<<<<<<<< taxonC
         ?<<<<<<<<<?                  ?<<<<<<<<        <<<<<<<< taxonD
                   >        ?<<<<<<<<<?        ?<<<<<<<<<<<<<<< taxonE
                   ?<<<<<<<<?         ?<<<<<<<<<<<<<<<<<<<<<<<< taxonF
                            ?<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< taxonG
```

7.5 Clearing trees from memory

Use the ClearTrees command to empty the tree buffer.

Clearing trees stored in memory:

Trees > Clear Trees

Ordinarily you will be prompted for conformation. If you specify NoWarn = Yes, the tree buffer is silently emptied.

7.6 Getting information about current trees

Use the TreeInfo command to obtain information on the status of trees currently in memory. No options are available.

Getting information about trees stored in memory:

Trees > Tree Info > Brief Status

7.7 Consensus Trees

Consensus trees are hierarchical summaries of the information common to a set of "rival" trees. PAUP* provides the following consensus tree methods: strict, semistrict (a variation on strict), Adams, and majority-rule. These methods differ in the criteria used for retaining ***groups*** (subsets) of taxa in the consensus. Introductory descriptions of consensus methods can be found in Swofford (1991) and Farris (1991).

NOTE: *For all consensus methods other than that of Adams (1972), if (1) the rival trees are unrooted, (2) outgroup-rooting is in effect, and (3) there are multiple outgroup taxa, then PAUP* first computes an unrooted consensus and then outgroup-roots the consensus tree. (Algorithmically, this is accomplished by rooting the tree at an arbitrarily chosen terminal taxon and applying the usual rooted consensus methods). As long as the specified ingroup taxa always comprise a monophyletic group, this approach is exactly equivalent to outgroup-rooting the rival trees first and then computing a rooted consensus. However, when the tree cannot be rooted such that the ingroup is monophyletic, the position of the root is not well specified, and the resulting consensus would depend on the method used to resolve the ambiguity in locating the root. The approach used in PAUP* circumvents this problem-the uncertainty in locating the root does not come into play until after the consensus is computed. This approach can not be used for the Adams (1972) method, which is defined only for rooted trees; consequently trees are rooted prior to calculation of the consensus in this case.*

In general, a consensus tree for a group of minimal trees will be longer than any of those trees (e.g., Miyamoto, 1985). This is because consensus trees are usually less-resolved than any of the rival trees, which automatically means that some characters will have to change more times on the unresolved consensus than they would have to on the more highly resolved rival trees. For example, the figure below shows two rival trees of length four. Their strict consensus is tree C. Because this tree is unresolved, both characters two and three must have two steps. Its total length is then five steps, one step longer than either of the rival trees.

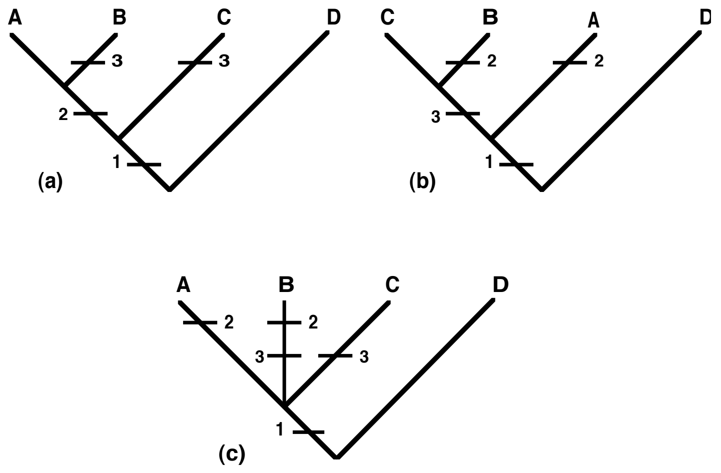


Figure 7.2 *Tree lengths of rival trees and their consensus. (a) rival tree of length four. (b) a different rival tree of length four. (c) strict consensus of (a) and (b), length is now five.*

The main point here is that the *consensus tree is not an optimal tree for a particular data set*, and should not be treated as such. It is a summary of agreement among trees, but it should not be interpreted as a phylogenetic tree. Such an interpretation assumes that polytomous nodes indicate simultaneous divergence of multiple lineages rather than uncertain resolution [i.e., "hard polytomies" vs. "soft polytomies; Maddison (1989)].

7.8 Strict Consensus

Strict consensus trees contain only those groups appearing in all of the rival trees (Sokal and Rohlf, 1981; Page, 1989). This can be considered to be the most conservative estimate of consensus, and is the simplest to interpret. In the figure below, tree f is the strict consensus of trees a, b, and c. It preserves the two groups (ABC) and (DEF), but there is no resolution within either of those groups. In the case of (EDF), two of the rival trees (a and c) agree on the resolution of (D(E,F)), while the other tree (b) resolves this clade as (E(D,F)). This conflict is reflected in the strict consensus by the trichotomy (E,D,F). In the case of (ABC), because only one tree resolves this trichotomy and two do not, the strict consensus must remain unresolved. The liability of this technique stems from its demand for identity in each of the rival trees. For example, two trees may be identical except for the placement of a single taxon, yet may have a completely unresolved consensus. In this instance, the consensus is ”too strict” (Adams, 1986; Funk and Brooks, 1990), in that none of the agreement between the trees is preserved in the consensus when the unstable taxon is included.

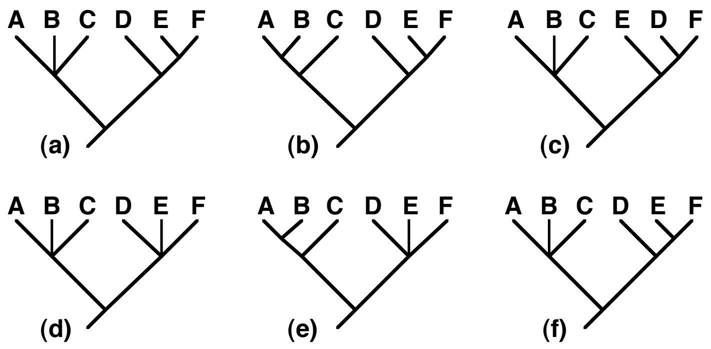


Figure 7.3 *Three rival trees (a,b,c) and their strict (d), semistrict(e) and 50% majority rule (f) consensus trees.*

7.9 Semistrict Consensus

This method corresponds to the ”combinable-component” consensus of (Bremer, 1990) [essentially the same idea was earlier proposed by Hillis (1987)]. Under this method, if all trees have either an (A,B,C) trichotomy or an ((A,B),C) dichotomy (i.e., $A + B$ is never contradicted, just not always supported), then (A,B) is retained in the consensus. In the example above, trees a and c are unresolved, while tree b supports $A + B$. The semistrict consensus (e) preserves $A + B$, while the strict consensus (f) does not. When there is conflict, semistrict behaves the same as strict. E.

7.10 Majority-rule consensus

In contrast to strict consensus, it may be of interest to find groups that appear on a certain pre-specified percentage of the rival trees. Thus a group may be preserved in the consensus even if there are some trees that support conflicting groups (see Margush and McMorris, 1981; Swofford, 1991). In the example above, the majority-rule consensus (d) preserves (EF) because it is found in two of the three rival trees (a and c; Figure 2.33). Likewise, (ABC) is unresolved because two of the three trees do not resolve it. Typically the majority-rule percentage is set to 50%, so that the consensus will retain all groups found in over half of the rival trees. The reason that groups must be present in over half the rival trees to be included is that two groups occurring in exactly half the trees might not be able to coexist on the same consensus tree. If only two trees are being compared, majority-rule and strict methods are equivalent.

7.11 Adams Consensus

The method of Adams (1972, 1986) (see also Swofford, 1991) was the first consensus method to be proposed. Adams trees typically preserve more structure than strict methods. The example shown below illustrates the behavior of Adams consensus. The strict consensus of trees a and b would be completely unresolved, as there are no groups that are exactly the same on both of the rival trees. The Adams consensus (tree c; Figure 2.34) shows more resolution, but it must be interpreted carefully. If this were a strict consensus, we would infer that there is a monophyletic group (ACDEF) on all of the rival trees, and that the relationships of B and G to that group are unresolved.

The Adams tree makes no such claim. Instead of being defined on the basis of monophyletic groups, it shows ”nestings” shared among the trees (one group is said to nest within a larger group if the most recent common ancestor of the smaller group is a descendant of the most recent common ancestor of the larger group, which need not require monophyly of either group). In this example, the most recent common ancestor of taxa A and C is a descendant of the most recent common ancestor of taxa A, C, and D on both rival trees; therefore (AC) is retained in the consensus. The Adams consensus reflects a great deal of shared structure between trees a and b, which are identical if taxa B and G are removed. It does not, however, imply that a monophyletic (ACDEF) is found on either of the rival trees. The appearance of B and G at the unresolved portion at the base of the consensus indicates that their placement relative to the other taxa is different on each of the rivals. It does not indicate that they both belong outside the ACDEF group, since the rival trees suggest that either B or G may be contained within that group.

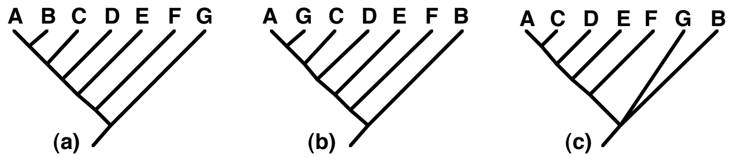


Figure 7.4 *Two rival trees (a, b) and their Adams consensus (c).*

Note that the group (AC) appears in the Adams consensus, yet is not found on either of the two rival trees (the same is true of (ACD), (ACDE), and (ACDEF)). This is a common occurrence with the Adams method, and has been the source of much of the criticism concerning it. This criticism is valid only if Adams trees are interpreted in the same way as are strict consensus trees. On a strict consensus, the appearance of an AC clade would indicate that clade appeared on all rival trees. On an Adams consensus, the appearance of an AC clade means instead that the rival trees indicate that A and C are more closely related to each other than either is to D, E, or F.

7.12 Consensus indices

Several authors have developed indices that use the topology of the consensus tree to measure the degree of congruence of the rival trees. Many of these measures are simple functions of the "resolution" of the consensus tree: the more congruent a set of rival trees is, the more highly resolved will be the consensus of those trees. Some indices are also affected by the symmetry of the tree. The consensus indices computed by PAUP* are described in Rohlf (1982) and Swofford (1991).

7.13 Calculating Consensus Trees

If multiple trees are in memory, you can calculate the consensus for those trees in four ways: Adams, strict, semistrict, and majority-rule. The details of each of the consensus methods are discussed in the section ”Consensus Trees.” You may calculate one or all of the different consensus types for any given set of rival trees-you do not have to issue repeated requests for each type. Once a consensus tree is calculated, it is not stored in the tree buffer. If you wish to use it as a constraint or perform other tree manipulations on it, you must first save it to a treefile using the Save option and then load it into memory as you would any other tree. In fact, you may wish to calculate consensus trees for different subsets of trees in memory and store them in separate files for further comparison.

Use the ConTree command to request computation of strict, semistrict (combinable component), Adams, and/or majority-rule consensus trees .

The tree list specifies which trees to include in the consensus; the default is ”All.”

Computing Consensus Trees:

Trees > Compute Consensus

Example 1:

The following example recalls bootstrap trees saved during three seperate bootstrap executions. See information regarding the GetTrees command. By including the UseTreeWts option, the ConTree command will display the bootstrap majority rule tree for the combined trees contained in the three seperate bootstap tree files.

Computing Consensus Trees:

- First, import all of the desired trees from their respective treefiles and apply tree weights (see ”Get trees from file” command)
- Select **Trees > Compute Consensus...**
- De-select the **Strict** option and check the **Majority-rule** box
- Select the **Use tree weights** option and click **OK**

The following 3 examples are done in the command-line of the display buffer in PAUP*.

Example 2:

```
contree 1 2 3/strict save file=summary.con;
```

will calculate the strict consensus for trees 1, 2, and 3 and save it in the file ”summary.con.” You may also specify Adams for an Adams consensus, MajRule for a majority-rule consensus, or SemiStrict for a semistrict consensus. Any or all of these may be specified on the same command line or on the same line of a Paup block.

Example 3:

Consensus indices will be printed following each tree if the Indices option is included.

```
contree 1 2 3/ strict adams indices file=summary.con;
```

will calculate and store the strict and Adams consensus trees and include consensus indices for both in the file ”summary.con.”

Example 4:

For majority-rule consensus trees, a table of partition or group frequencies can be requested, the percentage cutoff specified, and compatible trees occurring in less than fifty percent of the trees can be included by specifying

```
contree 1 2 3/majrule cutoff=50 dotplot le50;
```

The format of the partition-frequency table

```
Partitions found in one or more trees and frequency of
   occurrence:

12345678    Freq
----------------
...*****       3
...***..       3
..******       2
....**..       2
......**       2
```

This output is very useful if there is a very large number of trees in memory. If you are interested in all trees in which a particular group appears, it would be tedious to output all trees to find the ones of interest. A better strategy might be to compute a majority-rule consensus with partition-frequency table output; identify groupings of interest; construct a constraint tree that incorporates those groupings; and finally, filter the trees in memory to output only those trees on which the groupings occur.

7.14 Filtering Trees

Trees in memory can be selectively hidden by a process known as ”filtering”. The analogy is to a physical filter, which has certain characteristics that cause some elements to be retained, leaving a ”residue” on the filter itself. PAUP*’s filters are of this type, with the ”residue” being the trees that meet the filtering criteria. You build the filter by specifying its components, for example so that it will keep trees shorter or longer than a given length; within a certain topological distance from a reference tree; or trees compatible with a constraint tree. Unlike a physical filter, though, PAUP*’s filters can be inverted, that is, the ”residue” will be trees that do not meet the filter criteria. Filtering criteria are not cumulative-each time you filter trees the filtering criteria are applied to all undeleted trees in memory.

Use the Filter command to filter trees according to score, constraints, or other criteria.

If the Off option is specified, trees that do not satisfy the specified filtering criteria are retained. If the Purge option is specified, trees that are currently hidden by a tree filter will be deleted from memory.

Syntax: `Filter;`

Menu equivalent: **Trees > Filter**

Example 1:

In line one of the following example the Filter command retains trees with scores less than or equal to 500. The second line retains trees with scores greater than or equal to 500.

```
filter maxscore=500;
```

```
filter maxscore=500 reverse=yes;
```

Example 2:

```
filter maxlength=50 ;
```

There are several different filtering criteria you may select:

Filtering trees within three partition-metric units from a specified tree (tree 3) would be done by

```
filter sd=3 from=3;
```

Filtering with constraints would be invoked using the command

```
filter constraints=testcon
```

where the constraint-tree ”testcon” will be used. This can be either a backbone or monophyly constraint tree. Constraint trees must be defined before they can be invoked, as is the case when searching under constraints. Although several constraint trees may be defined, only one may be invoked in a particular filter.

You also have the option of keeping polytomous trees only if there are no other compatible resolutions of them in memory. For example, two resolutions of a trichotomy might receive potential support but the third resolution might have no conceivable character changes supporting it, in which case the zero-length branch would collapse to the trichotomy. In this case, a search will retain trees containing the two resolutions of the trichotomy as well as trees containing the unresolved trichotomy resulting from the third resolution. If you check the Do not retain a polytomous tree for which a more highly resolved compatible tree exists item by specifying the LessResolv option of the Filter command, e.g.:

```
filter lessresolv;
```

only the more highly resolved trees will be retained. Reversing this filter (see below) will keep the polytomous trees in lieu of more highly resolved versions of them.

Filters can be quite complex; several criteria can be applied at the same time, for example:

```
filter maxlength=50 constraint=testcon permdel;
```

would filter all trees below fifty steps that are compatible with the constraint tree ”testcon” and permanently delete them.

If you wish to permanently remove trees from memory which do not meet the filtering criteria, use the PermDel option. The command

```
filter constraints=test1 permdel;
```

would remove all trees not compatible with the constraint tree ”test1.” The set of trees in memory available to filter will then decrease. Deleted trees cannot be recovered - you must either repeat the search that produced them or reload them from a file, if one exists.

Using the Not option with the Filter command achieves the same result:

```
filter not constraints=testcon;
```

This will keep only trees that are not compatible with the constraint tree ”testcon.”

Use the RevFilter command to ”reverse” the effect of the current filter. All trees that are currently hidden by the filter will become visible, and all trees that were previously visible will be hidden. There are no options.

Syntax: `RevFilter;`

Menu equivalent: **Trees > Filter Trees > Reverse Filter**

When filtering is invoked, PAUP* will display the number of trees originally in memory and the number of trees retained by the filter. It doesn’t show the trees-you must do that yourself to see which trees survived the filter.

```
Trees filtered according to the following criteria:
    constraint tree = ’PAUP 2’

    Number of trees originally in memory = 17
    Number of trees retained by filter = 6
```

You can also filter trees directly by tree number using the From and To options of Filter or by selecting the tree range in the **Trees > Filter Trees** dialog box. Thus the command

```
   filter from=1 to=5
```

would retain trees one through five.

NOTE: *Filtering trees does not reset the tree numbers - the numbers are those assigned before the invocation of the filter.*

Removing the filter restores all undeleted trees. This is done by the **Trees >Filter Trees >Remove Filter** menu command or the command

```
   filter off;
```

7.15 Condensing Trees

PAUP* can either keep all dichotomous trees, whether or not there is evidence to resolve all branches, or it can collapse zero-length branches. This will have the effect of collapsing all branches for which there is no character support. After collapsing, duplicate trees are eliminated from the tree buffer (if a tree is collapsed to a resolution that already resides in the buffer, only one of those trees is retained). Remember that PAUP* will not allow trees to be condensed if this action would result in violation of the constraints for a tree that would otherwise satisfy them. Collapsing is the default, but it specified in the search dialog box or by using the Collapse option during a search, such as

```
hsearch addseq=asis swap=nni hold=1 collapse;
```

Use the Condense command to collapse zero-length branches into polytomies for all trees and then keep only those trees that are unique after the collapsing is accomplished.

Syntax: `Condense [options];`

Menu equivalent: **Tree >Condense Trees** ...

Available Options:

- **Collapse = No| MaxBrLen| MinBrLen| AmbEqual**
- **DelDupes = Yes| No**

Description of options

Collapse = No| MaxBrlen| MinBrlen| AmbEqual

Unless Collapse = No is specified, branches are collapsed according to the specified criterion. MaxBrlen is the method used in PAUP* (and Hennig86). MinBrlen collapses a branch if it is possible for it to have zero length. AmbEqual collapses a branch if the MprSets of the two incident nodes are identical. These latter two methods were introduced in Goloboff's NONA program. We do not recommend their use, but they are available for those who wish to use them.

DelDupes = Yes|No

Unless DelDupes is specified, duplicate trees will be eliminated.

7.16 Agreement Subtrees

Use the Agree command to calculate agreement subtrees.

Syntax: `Agree [tree-list] [/options];`

Menu equivalent: **Trees >Agreement subtrees**...

Available Options:

- **All = No| Sets| Trees| Both**
- **ShowTree = Yes| No**

- **TreeFile** = ***tree-file-name***
- * **Replace = Yes| No**
- * **Append = Yes| No**
- **TCompress = Yes| No**

*Option is nonpersistent

Description of options

All = No|Sets|Trees|Both

Use all to calculate agreement subtrees for the trees specified in the Agree [tree-list]. If All = Sets, a list of the agreement sets and the taxa included within each set is given. If All = Trees, all of the agreement subtrees are printed, overriding ShowTree = No if this option was specified.

ShowTree = Yes|No

Shows the agreement subtree(s) given in the [tree-list].

TreeFile = *tree-file-name*

If TreeFile is specified, a description of all agreement subtrees is output to a file containing a NEXUS-format Trees block. This option is not persistent; you must specify it on every Agree command for which you want a tree file to be saved. See "Input/Output files" for instructions to specify the full path name of a file.

The following options apply only if File is specified

Replace = Yes|No

Append = Yes|No

If the specified file already exists, you will be prompted for confirmation that the existing file should be replaced. Explicit specification of Replaces suppresses this prompt; the existing file will be quietly overwritten by the new data. Alternatively, you may specify Append, in which case a new Trees block will be concatenated to the end of an existing file.

Tcompress = Yes|No

See "Tree output options" .

7.17 Diagnosing Trees

Use the ShowTrees command to request a diagram of one or more trees with no other information (see also the Describe command).

Syntax: `ShowTrees [tree-list] [/options];`

Menu equivalent: **Trees >Show Trees** ...

Available Options:

- **TCompress = Yes|No**
- **ShowTaxNum = Yes |No**
- **Root = Outgroup|Lundberg|Midpoint**
- **OutRoot = Polytomy|Paraphyl|Monophyl**
- **UserBrLens = Yes|No**

Description of options

UserBrLens = Yes|No

If UserBrLens = Yes, then user-supplied branch lengths are used.

ShowTaxNum = Yes|No

When ShowTaxNum is set to Yes trees are displayed with both taxa labels and taxa numbers. The number are based on the order in which the taxa appear in the Characters or Data blocks.

See "Options And Subcommands Affecting Multiple Commands" for a description of the following options:

- TCompress = Yes|No
- Root = Outgroup|Lundberg|Midpoint
- OutRoot = Polytomy|Paraphyl|Monophyl

Similar to the ShowTree command, you can use the DescribeTrees command to output tree diagrams **and** associated information. Information regarding tree length is given according to the currently selected optimality criterion.

Syntax: `DescribeTrees [tree-list] [/options];`

The *tree-list* specifies the numbers of the trees you wish to describe. If this is the first DescribeTrees command and you do not specify a tree list, only the first tree is described.

Menu equivalent: **Trees >Describe Trees** ...

Available Options:

- **Plot = None| Cladogram| Phylogram| Both**
- **Root = Outgroup | Lundberg | Midpoint**
- **OutRoot = Polytomy| Paraphyl| Monophyl**
- **BrLens = Yes| No**
- **LabelNode = Yes| No**
- **XOut = None| Terminal| Internal| Both**
- **MPRsets = Yes| No**
- **ApoList = Yes| No**
- **ChgList = Yes| No**
- **Patristic = Yes| No**
- **Homoplasy = Yes| No**
- **FValue = Yes| No**
- **Diag = Yes| No**
- **TCompress = Yes| No**
- **CMLabels = Yes | No**
- **CMShowEq = Yes| No**
- **CMColWid = integer-value**
- **CMCStatus = Yes| No**

Description of options

Plot = None|Cladogram |Phylogram|Both

If Plot = Cladogram, branch lengths on the tree have no meaning, and taxa are aligned at the right edge of the diagram. If Plot = Phylogram, branch lengths are drawn proportionally to the number of changes assigned to each branch. Plot = Both requests output of the tree diagram in both Cladogram and Phylogram formats.

BrLens = Yes|No

Requests output of a table of assigned, minimum-possible, and maximum-possible branch lengths.

ChgList = Yes|No

Requests output of a list of changes in each character.

ApoList = Yes|No

If set to ApoList = Yes a list of the apomorphic characters is displayed. The list includes the characters that have changed along each branch, the consistency index for each character, and the original and transformed character. The double arrow "==>" under the change column represents unambiguous changes and the single arrow "–>" represents ambiguous changes.

Diag = Yes|No

Requests output of character diagnostics, including the minimum-possible, assigned, and maximum-possible length of each character, and goodness-of-fit measures based on these quantities.

Patristic = Yes | No

Requests output of the patristic distance matrix.

Homoplasy = Yes|No

Requests output of the pairwise homoplasy matrix.

LabelNode = Yes|No

Ordinarily, internal nodes on the tree diagram are labeled with a node number that is referenced by other output information (change lists, apomorphy lists, etc.). LabelNode = No can be used to suppress the labeling of internal nodes.

XOut = None|Terminal|Internal|Both

Requests output of a table of character-state assignments for each tree. XOut = Internal requests output of the character-states assigned to internal nodes for each tree. XOut = Terminal requests a listing of the original data matrix. (This option is not particularly useful, as you would ordinarily use the ShowMatrix command to list the data matrix.) XOut = Both requests a listing of the original data matrix plus the states assigned to internal nodes.

MPRSets = Yes | No

Requests a listing of the possible character-state assignments (MPR-sets) for each tree (parsimony criterion only).

FValue = Yes|No

Requests output of the F-value and F-ratio.

See "Options And Subcommands Affecting Multiple Commands" for details.

- Root = OutGroup|Lundberg|Midpoint
- OutRoot = Polytomy|Paraphyl|Monophyl
- TCompress = Yes|No
- CMLABELS = Yes|No
- CMShowEq = Yes|No
- CMCStatus = Yes|No
- CMColWid = *column-width*

7.18 Sorting trees

Use the SortTrees command to sort trees that are currently in memory according to the current optimality criterion.

Syntax: `SortTrees;`

Menu equivalent: **Trees >Sort Trees**

7.19 "Cladograms" and "Phylograms"

Trees output by the Describe command may be drawn either as cladograms or phylograms. Cladograms convey only branching information; phylograms draw branch lengths proportional to number of inferred changes according to the current reconstruction. This means that cladograms will not change when a different reconstruction is chosen, but phylograms will.

Example of a phylogram, obtained by

```
describe 1/ plot=phylogram
```

```
?<<<<<<<<< taxon1
>
> ?<<<<<<<<<<<< taxon2
> >
> >                 ?<<<<<< taxon3
?12                 >
  >                 >                             ?<<<<<<<<<<<<< taxon4
  >                 >                       ?<<<<<8
  ?<<<<<<<<<<<<<<<<11                       >     ?< taxon6
                    >           ?<<<<<<<<<<<9
                    >           >           ?<<<<<<<<<<<<<< taxon7
                    ?<<<<<<<<<<10
                                ? taxon5
```

The command

```
describe 1/ plot=cladogram
```

gives a cladogram for the same taxa:

```
?<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< taxon1
>
>          ?<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< taxon2
>          >
>          >           ?<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< taxon3
?<<<<<<<<<12           >
           >           >                                 ?<<<<<<<<<<< taxon4
           >           >                     ?<<<<<<<<<<<8
           ?<<<<<<<<<<11                     >           ?<<<<<<<<<<< taxon6
                       >          ?<<<<<<<<<<9
                       >          >          ?<<<<<<<<<<<<<<<<<<<<<<< taxon7
                       ?<<<<<<<<<10
                                  ?<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< taxon5
```

7.20 Zero-Length Branches and Polytomies

As discussed above, PAUP*'s search algorithms find only binary trees. When the data are insufficient to determine fully resolved trees, trees containing polytomies may be preferable to binary trees containing one or more branches of zero length. Dichotomies that collapse into polytomies when zero-length interior branches are deleted are said to represent **arbitrary resolutions**, because character support for some of the groupings is absent or ambiguous. Usually (but not always), a smaller number of trees containing one or more polytomies is preferable to the full set of binary trees; all of PAUP*"s search algorithms therefore provide an option for collapsing interior branches of zero length.

Unfortunately, however, the criterion for collapsing zero-length branches is complicated by the fact that for some characters, more than one **most parsimonious reconstruction** [**MPR**; Swofford and Maddison (1987)] may exist. An MPR is the set of state assignments to each interior node that allow the length required by a character on a given tree to be minimized. Consequently, a branch may have zero length under one MPR but a length greater than zero under a different MPR. Swofford and Maddison (1987) described an algorithm for finding the minimum and maximum lengths of a branch over all MPRs for a single character. We can obtain the "minimum possible" and "maximum possible" branch lengths by summing, over all characters, the minimum and maximum branch lengths for each character. Conceivably, three rules could then be used to decide whether to collapse an interior branch:

- Collapse an interior branch if the *minimum* possible length of the branch is zero. That is, if there exists at least one MPR for every character such that no length is assigned to the branch, the branch is collapsed.
- Adopt an ancillary criterion for choosing one MPR from the full set of MPRs for each character (or choose one arbitrarily). If no length is assigned to the branch for all characters, then the branch is collapsed.
- Collapse an interior branch if the maximum possible length of the branch is zero. That is, if all MPRs assign zero length to the branch for every character, then the branch is collapsed.

I believe that only the third criterion can be justified and this rule is the one implemented in PAUP*. (Comparisons of results from Farris's Hennig 86 program suggest that this rule is used by that program as well, although his documentation does not address the issue.) The rationale is simply that if potential, though perhaps ambiguous, support exists for a particular resolution of a polytomy, then the resolution should be retained. However, if no support exists for any resolution (i.e., a branch length is zero under every possible reconstruction), then a polytomy is in order. For example, if three trees found by PAUP* consist of a trichotomy and two of the three resolutions of that trichotomy. This means that there is potential support for two of the three resolutions of the trichotomy, but for the third resolution there are no characters that provide even potential support. Note that even when a branch is not collapsed, it may still be assigned zero length under some reconstructions. Thus, if you have requested the option to collapse branches and yet zero-length branches occur in the output, you will know that there are other MPRs in which the branch has nonzero length.

The first criterion is flawed in that it may not be possible to collapse all branches whose minimum length is zero and still obtain an MPR. For instance, two branches may each potentially have zero length, but once one of the branches is reduced to zero length, the other cannot be, and vice versa. An example is shown below. Two MPRs exist (A and B) for the hypothetical discrete character on the binary tree shown, requiring two steps. It can be seen from these reconstructions that all three of the branches may potentially be assigned no changes (zero length), but only the interior branch indicated by a heavy line in tree C has zero length under both reconstructions. If we collapse the zero-length branches under reconstructions ?a and ?b, we obtain the trees and reconstructions of trees D and E, respectively, which also require two steps. However, if all branches that are potentially of zero length (i.e., of zero length in at least one MPR) are collapsed, an MPR on the resulting tree (F) would then require three steps whether state 0 or state 1 were assigned to the single remaining internal node.

This example also illustrates why PAUP* uses the third criterion listed above rather than the second. If branches having zero length under *either* reconstruction ?a or reconstruction ?b (but not both) are collapsed, then either the group (A,B) or the group (C,D) would be retained. Clearly, the decision to retain one but not both of these groups would be arbitrary. PAUP* retains both groups, since there is potential support (i.e., favored by at least one reconstruction) for both. However, the group (A,B,C,D) is not retained since neither reconstruction provides even potential character support for it. Of course, some other character might provide support for the (A,B,C,D) group, and if so, the group would be retained. Otherwise, the tree would collapse as shown in tree G.

7.21 Table of Branch Lengths and Linkages

When a Table of linkages in the Describe Trees dialog box or the Links option is specified, PAUP* outputs a table which shows, for each terminal taxon and each node, which node it is linked to, the assigned length of the branch under the current reconstruction, and the maximum and minimum possible lengths of that branch. over all MPRs (see Swofford

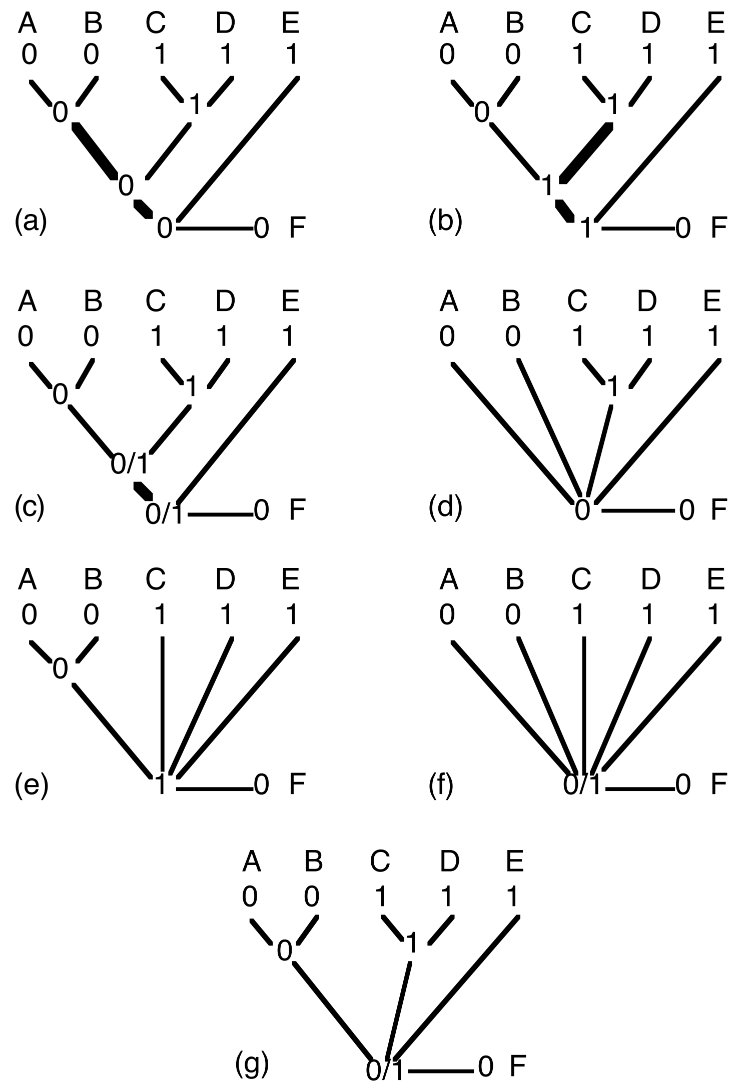


Figure 7.5 *Demonstration of effect of different criteria for collapsing zero-length branches. (a,b) The two MPRs for a hypothetical character on a given tree; heavy lines indicate zero-length interior branches. (c) Possible state assignments to interior nodes consistent with at least one MPR; heavy lines indicates mandatory zero-length interior branch. (d) Reconstruction A after collapsing all zero-length interior branches. (e) Reconstruction B after collapsing all zero-length interior branches. (f) Tree after collapsing all interior branches that are of zero length under at least one MPR. (g) tree after collapsing all branches that have zero length under all MPRs, retaining groups with potential (but ambiguous, in this case) support.*

and Maddison, 1987). The command format would be

```
describe 1/links;
```

Below is the format of the output. The terminal taxa are listed first, followed by the internal nodes.

```
Branch lengths and linkages for tree #5 (unrooted)


                                  Assigned       Minimum       Maximum
                   Connected        branch      possible      possible
   Node             to node         length        length        length
taxon1 (1)             12            2.000         2.000         4.000
taxon2 (2)             12            2.000         0.000         2.000
taxon3 (3)             11            1.000         0.000         1.000
taxon4 (4)              8            3.000         2.000         4.000
taxon5 (5)             10            0.000         0.000         1.000
taxon6 (6)              9            4.000         1.000         4.000
taxon7 (7)              8            3.000         2.000         4.000
    8                   9            1.000         1.000         2.000
    9                  10            2.000         2.000         4.000
   10                  11            2.000         2.000         3.000
   11                  12            3.000         2.000         3.000
```

7.22 Change and Apomorphy Lists

There are two ways of summarizing the character changes that have taken place on a particular tree. The first is to describe the changes for each character, the second for each branch. If List of changes is selected in the **Describe Trees** dialog box or the command `describe chglist` is issued, the output will show for each character, where on the tree that character changed, the state it changed from, and the state it changed to. If list of apomorphies is selected in the **Describe Trees** dialog box or Describe ApoList is requested, the output will show for each branch, which characters changed, the state they changed from, and the state they changed to.

The following command will give character changes:

```
describe 1/chglist;
```

with the following output:

```
Character change lists:
```

```
Character          CI     Steps         Changes

1                1.000      1     node_12 0 ==> 1 node_11
2                0.500      1     node_12 1 <=> 0 taxon1
                            1      node_8 1 ==> 0 taxon7
3                0.500      1     node_12 0 ==> 1 node_11
                            1      node_8 1 ==> 0 taxon6
4                1.000      1     node_11 0 ==> 1 node_10
5                1.000      1     node_10 0 ==> 1 node_9
6                0.500      1     node_10 0 ==> 1 taxon4
                            1      node_8 0 ==> 1 taxon7
7                0.333      1     node_12 0 <-> 1 taxon1
                            1     node_11 0 --> 1 node_10
                            1      node_8 1 ==> 0 taxon6
8                0.500      1     node_10 0 ==> 1 taxon4
                            1      node_8 0 ==> 1 taxon6
9                0.667      1     node_12 0 ==> 2 taxon2
                            1      node_9 0 ==> 1 taxon4
                            1      node_8 0 ==> 2 taxon7
10               1.000      1     node_11 0 ==> 2 node_10
                            1      node_9 2 ==> 1 node_8
```

You can see the arrows are not all the same type. This is because PAUP* uses different arrows to indicate different types of change. A double-lined arrow means that the change occurs in all possible reconstructions (i.e. is unambiguous). A single-lined arrow indicates that change occurs under some reconstructions but not others. A double-headed arrow indicates that the direction of change is undetermined, that is the change occurs along the branch connecting the outgroup to the ingroup. A single-headed arrow indicates that the direction of change is unambiguously within the ingroup. In the example above all four types of arrow are present. Remember that these arrows are a reflection of the tree and optimization criterion. Character changes with double-lined arrows will not be affected by a different optimization scheme, while those with single-lined arrows will. This is also true of the ouput listing the apomorphies for each branch, as below. Note that multistate characters have a CI of 1.000 when each state is derived only once.

To get the list of apomorphies for each branch, type:

```
describe 1/apolist;
```

This gives the following output:

```
Apomorphy lists:
```

```
              Branch                Character     Steps    CI  Change

        taxon1 <-> node_12          2                 1  0.500 0 <=> 1
                                    7                 1  0.333 1 <-> 0
                                    9                 1  1.000 0 <=> 1
                                    12                1  1.000 0 <-> 1
       node_12 --> node_11          1                 1  1.000 0 ==> 1
                                    3                 1  0.500 0 ==> 1
                                    12                1  1.000 1 --> 2
       node_11 --> node_10          4                 1  1.000 0 ==> 1
                                    7                 1  0.333 0 --> 1
       node_10 --> taxon4           6                 1  0.500 0 ==> 1
                                    8                 1  0.500 0 ==> 1
                                    12                1  1.000 2 ==> 4
       node_10 --> node_9           5                 1  1.000 0 ==> 1
        node_9 --> node_8           11                1  1.000 0 ==> 1
                                    12                1  1.000 2 ==> 3
        node_8 --> taxon6           3                 1  0.500 1 ==> 0
                                    7                 1  0.333 1 ==> 0
                                    8                 1  0.500 0 ==> 1
        node_8 --> taxon7           2                 1  0.500 1 ==> 0
                                    6                 1  0.500 0 --> 1
                                    10                1  1.000 1 --> 0
```

7.23 Character Diagnostics

In addition to information about where the characters change on a particular tree, PAUP* will display summary diagnostic information for each character. This is obtained by selecting **Character diagnostics** in the **Describe Trees** dialog box or the Describe command with the Diag option. The output includes the minimum possible length for each character (i.e. the length if a minimal tree were computed for each character taken separately); the maximum possible length (i.e. the length on a completely unresolved bush); the length required on the tree being described; and four goodness-of-fit statistics: the unit consistency index (CI); homoplasy index (HI); retention index (RI); and the rescaled consistency index (RC). This output is obtained by the following command:

```
describe 1/diag;
```

and has the following format:

```
Character diagnostics:

                Minimum     Tree  Maximum
Character         Steps    Steps    Steps        CI       HI       RI       RC

 1                    1        1        2     1.000    0.000    1.000    1.000
 2                    1        2        3     0.500    0.500    0.500    0.250
 3                    1        2        3     0.500    0.500    0.500    0.250
 4                    1        1        3     1.000    0.000    1.000    1.000
 5                    1        1        4     1.000    0.000    1.000    1.000
 6                    1        2        3     0.500    0.500    0.500    0.250
 7                    1        2        4     0.500    0.500    0.667    0.333
 8                    1        2        2     0.500    0.500    0.000    0.000
 9                    1        1        1     1.000    0.000      0/0      0/0
 10                   1        1        2     1.000    0.000    1.000    1.000
 11                   1        2        3     0.500    0.500    0.500    0.250
 12                 n/a        0        0       0/0      0/0      0/0      0/0
 13                 n/a        0        0       0/0      0/0      0/0      0/0
 14                   1        2        2     0.500    0.500    0.000    0.000
 15                   1        2        2     0.500    0.500    0.000    0.000
 16                   1        2        2     0.500    0.500    0.000    0.000
```

Note that these minimum and maximum lengths are not the same as those generated by the table of branch lengths and linkages, which only consider MPRs. The retention index is undefined for uninformative (autapomorphic) characters - this is indicated in the matrix by 0/0. The consistency index is not calculated for stepmatrix characters, due to the difficulty in estimating the minimum amount of change for those types of characters.

7.24 Character-State Reconstructions

PAUP* will also display reconstructed character states for the internal nodes of a tree, according to the optimization criterion currently in effect. It will also display the possible reconstructions over all MPRs. In this way you can compare the reconstructions required by the current tree and optimization with the states possible over all most parsimonious reconstructions.

There are several ways to obtain this information: assigned character changes are displayed superimposed on the current tree(s) when ChgPlot or character changes selection in the **Describe Trees** dialog box is selected. Selected reconstructions will be output with a tree for each character. States for the terminal taxa are also displayed.

NOTE: *If no characters are specified for either a* ChgPlot *or* PossPlot *command, the characters are taken to be those plotted in the last invocation of either of these commands. For example,* ChgPlot *1 3 5 7;* PossPlot *will cause both commands to output information for characters 1, 3, 5, and 7.*

The trees below are the reconstruction of a single character using the command

```
   describe 1/ chgplot;
```

The first tree optimized the character using ACCTRAN; the second used DELTRAN. You can see how different the two hypotheses are for character 1. The ACCTRAN hypothesis proposes an origin at the node leading to taxa 4,5,6, and 7, with a subsequent loss in taxon 5. The DELTRAN hypothesis prefers two independent origins of character 1: once in taxon 4 and once in taxon 7.

```
Changes in character 1 on tree 1:

?<<<<<<<<<<<<<<<<<<<<<<<<<
>          ?<<<<<<<<<<<<<<<<<<<<<<
?<<<<<<<<<<0          ?<<<<<<<<<<<<
           ?<<<<<<<<<<0          ?<<<<<<<<<
                      ,,,,,,,,,,,1          ,,,,,,,,
                                 ?<<<<<<<<<<1          ?<<<<<
                                            ?<<<<<<<<<<1<<<<<<


Changes in character 1 on tree 1:

?<<<<<<<<<<<<<<<<<<<<<<<<<
>          ?<<<<<<<<<<<<<<<<<<<<<<
?<<<<<<<<<<0          ?<<<<<<<<<<<<
           ?<<<<<<<<<<0          ,,,,,,,,,,
                      ?<<<<<<<<<<0          ?<<<<<<<
                                 ?<<<<<<<<<<0          ?<<<<<
                                            ,,,,,,,,,,,1<<<<<<
```

You can also display one tree summarizing the possible states for a given character. This is done using the PossPlot option of Describe command or Possible **state assignments** item in the **Describe Trees** dialog box. A typical command would be:

```
   describe 1/ possplot;
```

with the following output:

```
Possible state assignments for character 1 on tree 1:
?<<<<<<<<<<<<<<<<<<<<<<<<<
>          ?<<<<<<<<<<<<<<<<<<<<<<
?<<<<<<<<<<0          ?<<<<<<<<<<<<
           ?<<<<<<<<<<0          ?<<<<<<<<<
                      ?<<<<<<<<<<?          ?<<<<<<<
                                 ?<<<<<<<<<<?          ?<<<<<
                                            ?<<<<<<<<<<1<<<<<<
```

The two question marks indicate that the assignment of character 1 at those nodes is ambiguous - they do not indicate that the state is missing (default meaning of ”?” in a data matrix). Reconstruction of multistate characters may be ambiguous, or two or more states may be possible at a node, as below

```
Possible state assignments for character 3 on tree 2:

?<<<<<<<<<<<<<<<<<<<<<<<<<
>          ?<<<<<<<<<<<<<<<<<<<<<<
?<<<<<<<<<<0          ?<<<<<<<<<<<<
           ?<<<<<<<<<02          ?<<<<<<<<<<
                      ?<<<<<<<<<02          ?<<<<<<<<
                                 ?<<<<<<<<<<?          ?<<<<<
                                            ?<<<<<<<<<<1<<<<<<
```

You can also see the assigned and possible reconstructions by choosing states for interior nodes and possible state assignments selection in the **Describe Trees** dialog box or by using the XOUT option in the Describe command. The output describing the possible state assignments is a matrix of nodes by characters, as below. The possible states for each character is displayed in a row for each internal node on the tree. The output for the actual assignments for a given tree and optimization is also in matrix format. Requesting states for terminal taxa with the following command

```
   describe 1/xout=both
```

results in the printing of the input data matrix as well as the matrix of assigned states, as below

```
Data matrix and reconstructed states for internal nodes


                       1
Node         1234567890
----------------------
taxon1       0000001001
taxon2       0100000011
taxon3       1110000011
taxon4       1111100011
taxon5       1101100111
taxon6       1111011111
taxon7       1011111010
8            1111111011
9            1111100?11
10           111110?011
11           1110000011
12           0100000011
```

Once again, the question mark indicates that each available character state appears at that node in at least one reconstruction. It does not mean that any character is considered ”missing” at that point.

7.25 Stepmatrix Character Reconstruction: Special Considerations

When a stepmatrix contains three or more character states, it is possible that full minimization of the tree length may require assignments of character states that were not observed in any of the terminal taxa to internal nodes. When only a few character states were observed for a stepmatrix character, but the stepmatrix is defined for a large number of character states. This will often be the case for protein-sequences analyzed in a manner similar to the PROTPARS program for analyzing amino-acid sequences in PHYLIP. There, amino-acid sequences are analyzed using a stepmatrix specifying the cost of changing from one amino-acid to another (see the PROTPARS sample data matrix). However, not all amino-acids are always present in the terminal taxa. When this happens, execution in PAUP* is slowed significantly if all character-states are allowed as candidates for assignment to internal nodes.

PAUP* provides three options for dealing with this problem. First, you may choose to restrict the states for internal nodes only to those observed in terminal taxa. This is done using the StepMatrix = ObsOnly option of the Set command or selecting the appropriate box in the **Stepmatrices** dialog box. Second, you may choose to allow any state contained in the stepmatrix definition to be assigned to internal nodes regardless of whether it was observed in a terminal taxon. This is done using the StepMatrix = AllStates option of the Set command or the **Stepmatrices** menu command. For example, the following might be issued either in a PAUP* block or on the command line:

```
   set stepmatrix=allstates;
```

The third option provides a compromise between these two extremes. It uses the ”3+1” rule Applying the rule independently to each character, all triplets of observed character states are examined, in turn. For each triplet, each member of the triplet is assigned to one of the terminal nodes of a 3-branch tree. Then, every character-state contained in the stepmatrix definition is assigned, in turn, to the single internal node and the length required for the character, given these state assignments, is computed. If assignment of an unobserved character state to the internal node yields a length less than or equal to that obtained from one of the three members of the triplet, this unobserved state is added to the set of candidate states. This would be achieved by the command

```
   set stepmatrix=threeplus1;
```

How does the ”3+1” rule work in practice? Take the following stepmatrix for amino-acids as an example (this is taken directly from the PROTPARS example included with the PAUP* program - only the stepmatrix is reproduced here). Each of the symbols designates either one or more amino acids, a stop codon (*), or a deletion (-) corresponding to the standard IUB single-letter amino acid codes (see the section ”Predefined formats for molecular sequence data”).

```
     A C D E F G H I K L M N P Q R 1 2 T V W Y * -
[A]  0 2 1 1 2 1 2 2 2 2 2 2 2 1 2 2 1 2 1 1 2 2 2 3
[C]  2 0 2 2 1 1 2 2 2 2 2 2 2 2 2 1 1 1 2 2 1 1 1 3
[D]  1 2 0 1 2 1 1 2 2 2 2 1 2 2 2 2 2 2 1 2 1 2 3
[E]  1 2 1 0 2 1 2 2 1 2 2 2 2 1 2 2 2 2 1 2 2 1 3
[F]  2 1 2 2 0 2 2 1 2 1 2 2 2 2 2 1 2 2 1 2 1 2 3
[G]  1 1 1 1 2 0 2 2 2 2 2 2 2 2 1 2 1 2 1 1 2 1 3
[H]  2 2 1 2 2 2 0 2 2 1 2 1 1 1 1 2 2 2 2 2 1 2 3
[I]  2 2 2 2 1 2 2 0 1 1 1 1 2 2 1 2 1 1 1 2 2 2 3
[K]  2 2 2 1 2 2 2 1 0 2 1 1 2 1 1 2 2 1 2 2 2 1 3
[L]  2 2 2 2 1 2 1 1 2 0 1 2 1 1 1 1 2 2 1 1 2 1 3
[M]  2 2 2 2 2 2 2 1 1 1 0 2 2 2 1 2 2 1 1 2 3 2 3
[N]  2 2 1 2 2 2 1 1 1 2 2 0 2 2 2 2 1 1 2 3 1 2 3
```

```
[P]  1 2 2 2 2 2 1 2 2 1 2 2 0 1 1 1 2 1 2 2 2 2 3
[Q]  2 2 2 1 2 2 1 2 1 1 2 2 1 0 1 2 2 2 2 2 2 1 3
[R]  2 1 2 2 2 1 1 1 1 1 1 2 1 1 0 2 1 1 2 1 2 1 3
[1]  1 1 2 2 1 2 2 2 2 1 2 2 1 2 2 0 2 1 2 1 1 1 3
[2]  2 1 2 2 2 1 2 1 2 2 2 1 2 2 1 2 0 1 2 2 2 2 3
[T]  1 2 2 2 2 2 2 1 1 2 1 1 1 2 1 1 1 0 2 2 2 2 3
[V]  1 2 1 1 1 1 2 1 2 1 1 2 2 2 2 2 2 2 0 2 2 2 3
[W]  2 1 2 2 2 1 2 2 2 1 2 3 2 2 1 1 2 2 2 0 2 1 3
[Y]  2 1 1 2 1 2 1 2 2 2 3 1 2 2 2 1 2 2 2 2 0 1 3
[*]  2 1 2 1 2 1 2 2 1 1 2 2 2 1 1 1 2 2 2 1 1 0 3
[-]  3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 0

Stepmatrix for amino acid substitutions.
```

The stepmatrix gives the minimum number of amino acid replacement substitutions needed to convert one amino acid to another, based on the genetic code used in nuclear genes of most organisms and chloroplast genes in plants. The character distribution in the five taxa is:

```
      matrix
      Alpha      ABCDEFGHIK
      Beta       AB--EFGHIK
      Gamma      ?BCDSFG*??
      Delta      CIKDEFGHIK
      Epsilon    DIKDEFGHIK
      ;
```

where "?" is a missing observation and "-" is a deletion. Reconstructing the third position on one of the unrooted trees gives the following when only the observed states are allowed at internal nodes. Both "C" (cys) and "K" (lys) are possible assignments at the internal node:

```
Possible state assignments for character 3 on tree 1:

?<<<<<<<<<<<<<<<<<<<<<<<<<
>                      ,,,,,,,,,,,,,,,,,,,,,
C<<<<<<<<<<<<<<<<<<<<<CK                    ?
>                      ?<<<<<<<<<<<<<<<<<<<<
?<<<<<<<<<<<<<<<<<<<<<<<<<
```

Reconstructing internal nodes gives a different result when states that satisfy the "3+1" rule are allowed:

```
Possible state assignments for character 3 on tree 1:

?<<<<<<<<<<<<<<<<<<<<<<<<<
>                        ,,,,,,,,,,,,,,,,,,,,
C<<<<<<<<<<<<<<<<<<<CKR*                    ?<<
>                       ?<<<<<<<<<<<<<<<<<<<<
?<<<<<<<<<<<<<<<<<<<<<<<<<
```

Why are the additional states ″R″ (arg) and ″*″ (stop codon) allowed where only ″C″ (cys) and ″K″ (lys) were allowed in the previous example? The ″3+1″ rule takes all possible triplets of observed states and evaluates the length when every state in the stepmatrix is placed at the single internal node. Thus there is a ″3″ component composed of triplets of observed states, and a ″1″ component composed of each of the states in the stepmatrix. The figure below shows the results of the ″3+1″ rule when various are assigned as the ″1″ component. Since in this example there are only three observed character states, there is only one triplet (″C″, ″K″, and ″-″) to make up the ″3″ component. To begin, we measure the length when the members of the triplet (the observed states) are placed at the internal node. This is shown in a and b, where the overall length is 5 (the length to any state from the ″-″, or missing state, is always 3). Thus for any other state to pass the ″3+1″ rule, the length of the tree must be less than or equal to 5, which is the length obtained when only observed states are allowed. If the calculated length from the stepmatrix is more than 5, the state in question fails the ″3+1″ rule, and cannot be assigned to the interior node. In c, a state is determined to pass the rule (length=5, equal to that obtained by using the observed states) that is not one of the observed states. The length of tree d is 7, so state ″D″ fails the test and cannot be a candidate for assignment to the interior node.

For real data sets, I have never discovered a case for which StepMatrix = ThreePlus1 fails to obtain an optimal reconstruction as long as all triplets of states satisfy the triangle inequality. However, hypothetical counterexamples have been constructed (Wayne Maddison, personal communication). For example, the 3+1 rule excludes a state required to obtain an optimal reconstruction for the following stepmatrix.

```
        a  b  c  d e
a       -  13 13 13 9
b       13 -  13 13 9
c       13 13 -  13 9
d       13 13 13 -  9
e       9  9  9  9 -
```

If the only observed states are a, b, c, and d, the 3+1 rule will not allow

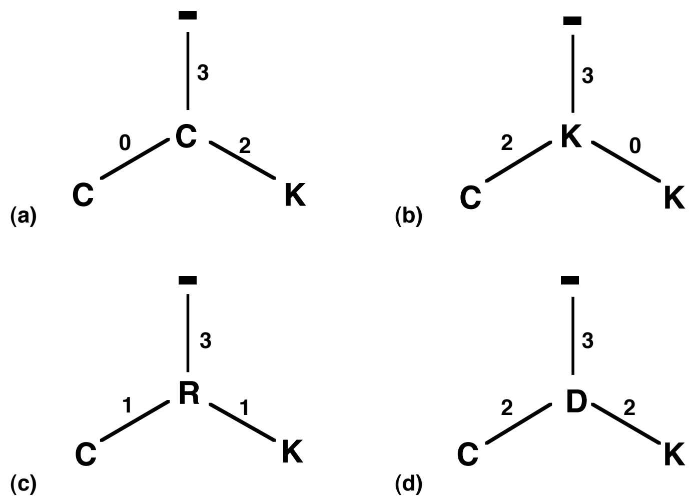


Figure 7.6 *Example of the "3+1" rule. (a) and (b) three-taxon trees with observed states at node, length=5. (c) non-observed state that satisfies the rule with length=5. (d) state which does not pass the rule, length=7.*

state "e" to be reconstructed at internal nodes. The cost of assigning any of the observed states will always be 26, while the cost of assigning e will always be 27, so the 3+1 rule excludes it as a candidate for internal nodes. However, for the following four-taxon tree, state e is actually the best state for internal nodes, even though it fails the 3+1 test. Assigning e gives a tree-length of 36 (=4 x 9), while assigning any other state gives a tree-length of 39 (3 x 13 plus 1 x 0).

Of course this example is derived from a fairly unusual stepmatrix. In reality the 3+1 rule will choose states that allow optimal reconstructions most of the time, and the speedup in searches can be enormous because fewer states need be considered.

7.26 The Pairwise Homoplasy and Patristic Distance Matrices

If you want to display a summary of homoplasy and patristic distances between taxa (where patristic distances (P) = sum of branch lengths on path between each pair of taxa; D = observed character difference; and the

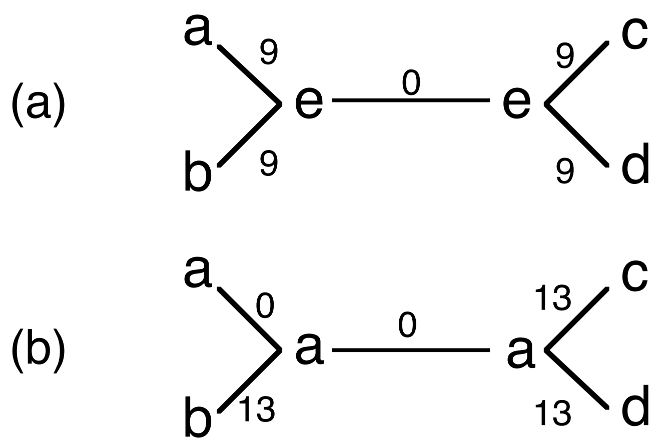


Figure 7.7 *Example of failure of 3+1 rule. (a) optimal reconstruction with length 36 not found by 3+1 rule. (b) reconstruction of length 39 allowed by 3+1 rule.*

homoplasy (H) = P - D), check the **Patristic distance** matrix and/or **Homoplasy matrix** items in the **Describe Trees** dialog box or use the commandHomoplasy or Patristic option of the Describe command. For example, the command

```
   describe 1/patristic homoplasy;
```

will output patristic distance and homoplasy matrices for the first tree:

```
Note: Multistate unordered and/or stepmatrix characters
     excluded from patristic distance calculations.


Patristic distance matrix

   Below diagonal: Adjusted character distances
   Above diagonal: Patristic distances

                    1         2         3         4         5
  1 taxon1          -         3         6         8         9
  2 taxon2          3         -         5         7         8
  3 taxon3          6         5         -         6         7
  4 taxon4          8         5         4         -         5
  5 taxon5          7         8         5         5         -
```

```
Pairwise homoplasy matrix

                   1         2         3         4         5
  1 taxon1         -
  2 taxon2         0         -
  3 taxon3         0         0         -
  4 taxon4         0         2         2         -
  5 taxon5         2         0         2         0         -
```

Stepmatrix characters and unordered multistate characters are not included in patristic distance calculations (and by extension, homoplasy matrices).

Although it is probably not a limitation in practice, you must choose costs and weights for stepmatrix characters such that the total length of the longest possible tree (a completely unresolved bush) is less than (231)-1 = 2,147,483,647.

7.27 Lengths and Fit Measures

Character information can be output in slightly different format by selecting LenFit or the **Lengths and Fit Measures** menu command. Typically, these options are used to display the lengths and fit statistics of trees in memory, but also to display summary information about characters relative to trees in memory. You can choose to output information about all characters (Single = All) or only those which vary over trees (Single = Var). This is a very good way to get a summary of the relative fit of various characters on different trees. You can choose to output any combination of the length, consistency index, retention index, or rescaled consistency index of characters. The output will include the value for each character over each chosen tree, as well as an entry for the best and worst value for that character. The following command

```
   lenfit 1-5/total single=all ci;
```

will, for trees one through five, output tree lengths and consistency indices for all characters (single=all), overall tree-lengths and ensemble consistency indices (total). If you wish only to output the range of minimum and maximum tree lengths (or best and worst fit measures) for each character, use the Range option.

For example, the command

```
lenfit/single=all total ci notl;
```

requests consistency indices (and suppression of tree lengths) for all characters as well as ensemble consistency indices:

```
   Sum of min. possible lengths = 14
   Sum of max. possible lengths = 36

Tree \#        1       2       3       4       5       6
---------------------------------------------------------
CI         0.609   0.609   0.609   0.609   0.609   0.609

Consistency indices for each character:


Tree           1       2       3       4       5       6       7       8
--------------------------------------------------------------------------
   1       1.000   0.500   0.500   1.000   1.000   0.500   0.500   0.500
   2       1.000   0.500   0.500   1.000   0.500   1.000   0.333   0.500
   3       0.500   0.500   0.333   0.500   1.000   0.500   0.500   0.500
   4       1.000   0.500   0.500   1.000   0.500   1.000   0.333   0.500
   5       1.000   0.500   0.500   1.000   0.500   0.500   0.500   1.000
   6       1.000   0.500   0.500   1.000   0.500   1.000   0.333   0.500
Best       1.000   0.500   0.500   1.000   1.000   1.000   0.500   1.000
Worst      0.500   0.500   0.333   0.500   0.500   0.500   0.333   0.500


Tree           9      10      11      14      15      16
----------------------------------------------------------
   1       1.000   1.000   0.500   0.500   0.500   0.500
   2       1.000   1.000   1.000   0.500   0.500   0.500
   3       1.000   1.000   0.500   1.000   1.000   1.000
   4       1.000   1.000   1.000   0.500   0.500   0.500
   5       1.000   1.000   0.500   0.500   0.500   0.500
   6       1.000   1.000   1.000   0.500   0.500   0.500
Best       1.000   1.000   1.000   1.000   1.000   1.000
Worst      1.000   1.000   0.500   0.500   0.500   0.500
```

If there are many characters in the data matrix, the most efficient use of this option is probably to limit output only to characters which vary over trees in memory (i.e., using the Single = Var option). That way you don't waste a lot of space describing characters which do not change over trees.

The output for LenFit does not give the detail available from a Describe command, in that it does not tell you where on the tree a particular character changed, only the amount of change that had to be invoked to explain that character on each tree. The advantage of using LenFit is that you can quickly see which characters or sets of characters are consistent with each tree. This can be very interesting if there are both minimal and non-minimal trees in memory. It may be that some characters are consistent only with topologies that are not found on minimal trees.

NOTE: *Minimum-possible tree lengths are not easily determined when multistate polymorphic taxa are present and the character type is Dollo. Consequently, character-fit measures are not evaluated in that case.*

7.28 Outgroups, Ancestors, and Roots

In most cases, trees computed by PAUP* are unrooted and need be rooted for output and interpretational purposes only. The absence of a defined root is a consequence of the fact that for undirected character types, the tree length is the same regardless of the position of the root. However, rooted trees are computed when any of the following are true: (1) characters of type "irreversible" are present; (2) asymmetric stepmatrix characters are present; or (3) a hypothesized ancestral taxon is explicitly included in the analysis at the request of the user.

For irreversible and asymmetric stepmatrix characters, the length of the tree depends upon the position of the root, so the trees must be stored as rooted trees. When all characters are undirected, the inclusion of a hypothesized ancestral taxon corresponds to the traditional, but unnecessary practice (Meacham, 1984, 1986) of *a priori* character polarization. This approach is reasonable when the polarity determination is unambiguous (i.e., there is no heterogeneity in the outgroup for characters that are variable within the ingroup). However, when the outgroup is heterogeneous, the most parsimonious assignment of an ancestral condition for the ingroup depends upon how the outgroup taxa are related to each other [see Maddison et al. (1984) for a complete analysis of this problem]. If outgroup relationships are already well resolved, you can use methods described by Maddison et al. (1984) to assess character polarities a priori and include an ancestral taxon in the analysis [see also Donoghue and Maddison (1986)]. Otherwise, it makes much more sense either to:

- include several outgroup taxa in the analysis, with the root of the ingroup portion of the tree being determined by the location at which the outgroup taxa connect to the tree [i.e., Maddison et al. (1984) ”simultaneous resolution of ingroups and outgroups taken to the extreme”; see also Farris (1972)], or

- compute an unrooted tree for the ingroup taxa only, making no a priori assumptions about polarity, and then attach an outgroup taxon (or hypothesized ancestor) to the tree a posteriori (Lundberg, 1972).

Although clearly better than ”guessing” at the polarity, both of the above approaches have some disadvantages (Maddison et al., 1984; Meacham, 1984; Donoghue and Maddison, 1986; Meacham, 1986). In order to be fully effective, the first method requires the inclusion of characters that are informative with respect to outgroup relationships even if these characters are invariant within the ingroup (Maddison et al., 1984). On the other hand, the second method can generate ”optimal” trees that are locally parsimonious within the ingroup, but not globally parsimonious.

If the trees being evaluated are rooted (either implicitly because directed characters are present or explicitly due to the user’s decision to include an ancestral taxon), the character states taken by the ancestor of the full tree must be specified. By default, these ancestral states are set to ”missing” (= the ”standard” ancestor), however, the user may make alternative assignments using AncStates definitions (see below). The default setting of all-missing values was chosen for two reasons. First, if all characters are undirected, trees may be converted from rooted to unrooted or vice versa with no change in tree length. Second, if directed characters are present, the program has the freedom to assign a character-state to the ”first fork” of the tree (i.e., the internal node immediately descending from the root corresponding to the initial branching event) that permits the most parsimonious reconstruction subject to the constraints imposed by the character type. Consider, for example, an irreversible character with polarity ”up.” If, for whatever reason, the state for all terminal taxa included in the analysis is ”1,” then the character will contribute no length to the tree, as expected for a constant character (A, below) But if the ancestral state had defaulted to ”0” rather than ”missing,” a step would have been assigned to the basal branch (B, below)

```
1   1   1   1   1   1
 \ /   /     \ /   /
  1   /       1   /
   \ /         \ /
    1           1
     \           \
      ?           0
```

A B

Reconstruction for a hypothetical character with ancestral state "missing" (a) and "0" (b). See text.

PAUP* provides commands for converting unrooted trees stored in memory to rooted trees and vice versa. Note that ***you do not have to invoke a "root trees" command*** before "showing," "describing" or "printing" trees, requesting consensus trees, etc. (see command descriptions below). The only time you would ordinarily use a "root trees" command is if you want to declare one or more characters to be irreversible or an asymmetric stepmatrix type when there are already unrooted trees in memory; in this case PAUP* must store the trees internally as rooted and you will have to convert them. Otherwise, the trees are stored as unrooted trees and are rooted automatically for output purposes (using either the currently defined outgroup or the Lundberg method, at your choosing).

Outgroup rooting of the unrooted tree A in the figure below can be done in three ways indicated by the three arrows leading to the rooted trees. "Pulling" the tree down at each of the different arrows leads to a different rooting: the tree can be rooted such that the outgroup is a monophyletic sister-taxon to the ingroup (tree B); it can be rooted such that the outgroup is paraphyletic with respect to the ingroup (tree C); or it can be rooted such that the outgroup taxa form a basal polytomy with the ingroup (Tree D; this is the default). The reason that taxon A and not taxon B is the basal-most taxon when the tree is rooted such that the outgroup is paraphyletic is that taxon A is the primary outgroup (the first outgroup taxon listed by the Outgroup command or chosen using the **Define Outgroup** menu command. The primary outgroup will always be the basal-most taxon when outgroup rooting and a paraphyletic outgroup is chosen.

You have three options when using outgroup rooting and more than one outgroup is present in the analysis. These are: rooting at an internal node with a basal polytomy; rooting such that the ingroup is monophyletic and the outgroup is paraphyletic with respect to the ingroup; or rooting such that the ingroup is monophyletic with the outgroup a monophyletic sister group. These options allow you to at least partially constrain the structure of the outgroup topology when the tree is rooted, providing the specified ingroup is monophyletic. See the section "Rooting trees for output and character reconstruction." If, however, the tree cannot be rooted such that the ingroup is monophyletic, PAUP* will include as many outgroup taxa as necessary to make the ingroup plus those outgroup taxa monophyletic. The primary outgroup is never included in

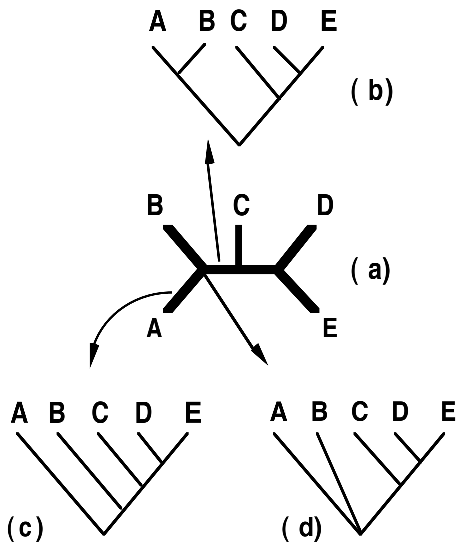


Figure 7.8 *Outgroup rooting options. (a) unrooted tree for five taxa, outgroup is A + B. (b) tree rooted with monophyletic outgroup. (c) tree rooted with paraphyletic outgroup. (d) tree rooted with basal polytomy.*

the ingroup, so the topology of the tree in that instance is directly affected by the choice of the primary outgroup.

7.29 Rooting Trees for Output and Character Diagnosis

The trees found by PAUP* searches are unrooted (unless there is some reason to explicitly root them in memory - see ”Outgroups, Ancestors, and Roots”). These trees must be ouput from the tree buffer as rooted trees for character reconstructions and other analyses. There are three rooting options - outgroup; Lundberg; and midpoint. Outgroup rooting, which includes an assumed outgroup in the analysis, is the most common method.

NOTE: *Remember that if you specify outgroup rooting but do not specify an outgroup, PAUP* will take the first taxon in the matrix as the default outgroup.*

Use the RootTrees command to convert all trees in memory from an unrooted to a rooted representation. Trees are rooted according to the currently specified outgroup.

Syntax: `RootTrees [options];`

Menu equivalent: **Trees >Root Trees**

Available Options:

- **OutRoot = Polytomy|Paraphyl |Monophyl**
- **Method = Outgroup|Lundberg|Midpoint**
- **UserBrLens = Yes|No**

Description of options

OutRoot = Polytomy|Paraphyl|Monophyl

See ”Options And Subcommands Affecting Multiple Commands” for details.

Method = Outgroup|Lundberg|Midpoint

The Method option is used to specify how unrooted trees are to be rooted prior to output. You can choose Outgroup rooting, using whichever outgroup you have selected; Midpoint rooting, which roots the tree at its midpoint; or Lundberg rooting, which requires that a previous AncStates command has been issued. By default, Outgroup rooting is in effect.

UserBrLens = Yes|No

If Method = Midpoint, then you may specify user-defined branch lengths.

Outgroup rooting is achieved by using the Root= Outgroup option, and perhaps specifying an Outgroup option as well. For example, the following command would describe a tree and root it such that the outgroup is monophyletic relative to the ingroup:

```
describe 1/ root=outgroup outroot=monophyl;
```

Other options for outgroup rooting output the outgroup in a basal polytomy (Outgroup = Polytomy) or as a group paraphyletic to the ingroup (Outgroup = Paraphyl). Remember that if the tree cannot be rooted such the specified ingroup is monophyletic, PAUP* will include as many outgroup taxa as possible with the ingroup to form a monophyletic group (though the primary outgroup is never included).

Lundberg rooting involves computing an unrooted tree for the ingroup taxa only, and then attaching an outgroup taxon (or hypothesized ancestor) to the tree a posteriori (Lundberg, 1972). This is achieved by using either the Root= Lundberg option or the appropriate selection in the **Rooting** dialog box. The tree can be output with Lundberg rooting with a command like the following:

```
describe 1/root=lundberg;
```

Lundberg rooting forces the use of MinF optimization (the only available optimization that is independent of the location of the root), which means that it may be invoked only when ordered (Wagner) or unordered characters are in use. Other character types (Dollo, irreversible, stepmatrix) cannot be used with MinF optimization; hence are not allowed. It is possible that a particular ancestor may have multiple equal-length attachment sites, which leads to the following output:

```
Note: Ties found in Lundberg rooting.  The following root
      locations are equally parsimonious:

      9 <--> 10
      14 <--> taxonA

Tree length = 23
Consistency index (CI) = 0.609
Homoplasy index (HI) = 0.391
Retention index (RI) = 0.591
Rescaled consistency index (RC) = 0.360
```

```
                                                       ?<<<<<<<<<<< taxonA
                                            ?<<<<<<<<<14<<<<<<<<<<< taxonB
                                 ?<<<<<<<<<13<<<<<<<<<<<<<<<<<<<<<< taxonC
                      ?<<<<<<<<<12<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< taxonF
           ?<<<<<<<<<11<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< taxonE
?<<<<<<<<<10<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< taxonD
>                                                      ?<<<<<<<<<<< taxonG
?<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< taxonH
```

You can see that PAUP* only prints one of the trees, but you may wish to recover all equally parsimonious trees for further analysis. To recover all the multiple roots for a optimal trees, you must include an ”ancestor” taxon in the matrix with the same states as were in the original AncStates definition (but do not change the AncStates settings). Then, exclude the ancestor and calculate all optimal trees for the ingroup. Next, save the trees to a file. Restore the ancestor. Load the saved trees as backbone constraints. Now perform a search while enforcing each constraint tree in turn and save all trees compatible with the backbone. Output them using Lundberg rooting-since the AncStates and ancestor are identical, you will be able to see each of the places the tree can be rooted. If there is only one possible rooting, a single tree will result from this process. Below are the two trees that are recovered for the example above. The first tree is identical to the tree displayed above.

```
Tree number 1:

                                                         ?<<<<<<<<< taxonA
                                               ?<<<<<<<<16<<<<<<<<< taxonB
                                      ?<<<<<<<15<<<<<<<<<<<<<<<<<<< taxonC
                            ?<<<<<<<<14<<<<<<<<<<<<<<<<<<<<<<<<<<<< taxonF
                   ?<<<<<<<13<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< taxonE
         ?<<<<<<<<12<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< taxonD
?<<<<<<<11                                               ?<<<<<<<<< taxonG
>        ?<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< taxonH
?<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< ancestor

Tree number 2:

        ?<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< taxonA
        >       ?<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< taxonB
?<<<<<<16       >        ?<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< taxonC
```

```
>       ?<<<<<<15        >                       ?<<<<<<<<<<<<<<<<< taxonD
>               ?<<<<<<<14               ?<<<<<<11        ?<<<<<<<< taxonG
>                        >       ?<<<<<<12       ?<<<<<<<10<<<<<<<< taxonH
>                        ?<<<<<<13       ?<<<<<<<<<<<<<<<<<<<<<<<<< taxonE
>                                ?<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< taxonF
?<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< ancestor
```

These trees can then be saved or analyzed further.

The last rooting option, Midpoint rooting, is straightforward: the tree is rooted at the midpoint of the longest path connecting any pair of taxa. The following command will output a tree with midpoint rooting:

```
   describe 1/root=midpoint;
```

Essentially, this is a way to output a tree without explicitly rooting it, and should only be used as a last resort. Use of this option forces the use of MinF optimization, which is the only optimization method that does not depend on the position of the root.

7.30 Derooting Trees

Normally, trees are stored unrooted in the tree buffer. They are automatically rooted when output is requested, based on the rooting criteria that are in effect. You need not actively request rooted trees unless you wish to define an ancestor for the full tree (see AncStates command) or if you are using directed, for example irreversible, characters. Trees are rooted according to the currently specified outgroup. There are no options. Once trees in memory are rooted, use the Deroot command to convert all trees in memory from a rooted to an unrooted representation. The circumstances under which you would need to use this command are rather unlikely.

Syntax: `DerootTrees;`

Menu equivalent: **Trees >Deroot Trees**

7.31 Comparing Trees

As we discussed in the context of ”islands” in the section on heuristic searching (add linked reference), there may be distinct families of trees that are similar with a family but very dissimilar to trees in other families (Hendy et al., 1988; Maddison, 1991). Random addition sequence searches provide one way to discover islands. Another way to examine this phenomenon is to use one of several metrics that measure the dissimilarity of pairs of trees. PAUP* provides three metrics, the symmetric-difference or "partition" metric (Penny and Hendy, 1985), which is equivalent to the Robinson and Foulds (1981) contraction/decontraction metric, the agreement metrics d and d1 described by Goddard et al. (1994). The symmetric difference metric simply counts the number of groups that are on one tree or the other but not on both. Trees that are most similar will have the lowest distance value. The output is potentially useful in identifying "classes" of similar trees.

Use the TreeDist command to request output of a matrix of tree-to-tree distances computed according to the symmetric-difference or "partition" metric, or one of two metrics based on agreement subtrees.

Syntax: `TreeDist [options];`

Menu equivalent: **Trees >Tree-to-Tree Distances** ...

By default, a matrix of pairwise distances between all trees is computed, as well as a frequency distribution of the distance values. You may choose to suppress one or both of the options. You can also choose to compare all trees in memory to a reference tree rather than comparing all pairs of trees. Here is a sample of the output (generated from the 25 most parsimonious trees for the "Menidia" sample data set).

Example:

```
   treedist;
```

with no options requests the full matrix of pairwise comparisons:

```
Symmetric-difference distances between trees


        1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24
1       -
2       4   -
3       2   4   -
4       1   3   1   -
5       3   7   5   4   -
6       2   6   4   3   3   -
7       3   7   5   4   2   3   -
```

```
8     1  5  3  2  4  1  2  -
9     7  3  7  6  4  7  6  8  -
10    6  2  6  5  7  4  7  5  3  -
11    7  3  7  6  6  7  4  6  2  3  -
12    5  1  5  4  8  5  6  4  4  1  2  -
13    6  2  6  5  9  8  9  7  5  4  5  3  -
14    3  5  1  2  6  3  4  2  8  5  6  4  7  -
15    2  4  2  1  5  2  3  1  7  4  5  3  6  1  -
16    7  3  7  6 10  7  8  6  6  3  4  2  1  6  5  -
17    5  7  3  4  2  5  4  6  4  7  6  8  9  4  5 10  -
18    5  7  3  4  4  5  2  4  6  7  4  6  9  2  3  8  2  -
19    4  6  2  3  5  2  5  3  7  4  7  5  8  1  2  7  3  3  -
20    4  6  4  3  1  4  3  5  3  6  5  7  8  5  4  9  1  3  4  -
21    4  6  4  3  3  4  1  3  5  6  3  5  8  3  2  7  3  1  4  2  -
22    9  5  9  8  6  9  8 10  2  5  4  6  3 10  9  4  6  8  9  5  7  -
23    8  4  8  7  9  6  9  7  5  2  5  3  2  7  6  1  9  9  6  8  8  3  -
24    9  5  9  8  8  9  6  8  4  5  2  4  3  8  7  2  8  6  9  7  5  2  3  -
25    3  5  3  2  4  1  4  2  6  3  6  4  7  2  1  6  4  4  1  3  3  8  5  8


Symmetric-difference distances between trees (continued)


     25
25    -


Distribution of symmetric-difference distances between trees


   ?<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
 0 > (0)
 1 >,,,,,,,,,,,,,,,,,,,,,,,,,,,,, (20)
 2 >,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,, (33)
 3 >,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,, (45)
 4 >,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,, (47)
 5 >,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,, (39)
 6 >,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,, (38)
 7 >,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,, (32)
 8 >,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,, (24)
 9 >,,,,,,,,,,,,,,,,,,,,,,,,,, (18)
10 >,,,,,, (4)
   ?<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
```

To limit the analysis to comparisons of one tree against all others, specify a tree number for the Compare option:

```
   treedist compare=1;
```

```
Symmetric-difference distances to tree 1

Tree number     1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21
Distance        -  4  2  1  3  2  3  1  7  6  7  5  6  3  2  7  5  5  4  4  4

Tree number    22 23 24 25
Distance        9  8  9  3

Distribution of symmetric-difference distances to tree 1


  ?<<<<<<<<<<<<<<<<<<<<<<<<
0 > (0)
1 >,,,,,,,,,,,,,,,,,,,,,,,,
2 >,,,,,,,,,,,,,,,,,,,,,,,,
3 >,,,,,,,,,,,,,,,,,,,,,,,,
4 >,,,,,,,,,,,,,,,,,,,,,,,,
5 >,,,,,,,,,,,,,,,,,,,,,,,,
6 >,,,,,,,,,,,,,,,,,,,,,,,,
7 >,,,,,,,,,,,,,,,,,,,,,,,,
8 >,,,,,,,,,,,,,,,,, (1)
9 >,,,,,,,,,,,,,,,,,,,,,,,,
  ?<<<<<<<<<<<<<<<<<<<<<<<<
```

Available Options:

- **Metric = SymDiff|AgD1|Agreement**
- **FromTree =** ***integer-value***
- **FD = Yes|No**
- **ShowAll = Yes|No**
- **File =** ***export-file-name***
- ***Replace = Yes|No**
- ***Append = Yes|No**

*Option is nonpersistent

Description of options

Metric = SymDiff|AgD1|Agreement

Use Metric to specific the metric used to calculate tree-to-tree distances. By default, the symmetric-difference metric (SymDiff) (Penny and Hendy, 1985) is used; however, you may also choose between two agreement-subtree metrics d (Agreement) and d1 (AgD1) Goddard et al. (1994).

FromTree = integer-value

If FromTree is specified, then the tree given by the integer-value will be compared to all other trees retained in memory. The default (FromTree = 0) specifies that all pairwise tree comparison be made.

FD = Yes|No

By default, the frequency distribution of tree-to-tree distances is output. Use FD = No to suppress this output, or FD to reverse the effect of a previous FD = No specification.

ShowAll = Yes|No

By default, a matrix of the tree-to-tree distances for all comparisons performed is output. Use ShowAll = No to suppress the output, or ShowAll to reverse the effect of a previous ShowAll = No specification.

File = *export-file-name*

Use the File option to save the tree-to-tree distances (as specified under the Metric option) to a file.

Replace = Yes|No

Append = Yes|No

If File = *export-file-name* option is specified and the *export-file-name* already exists, you will be prompted for confirmation that the existing file should be replaced. Explicit specification of Replace = Yes suppresses this prompt and the existing file will be quietly overwritten by the new data.

Alternatively, you may specify Append = Yes, in which case a new Trees block will be concatenated to the end of an existing file.

7.32 Matrix Representation Parsimony

Use dthe MatrixRep command to save trees in ”matrix representation.” A NEXUS file is created that contains a dummy character for each clade on the tree; analysis of the resulting data set under parsimony will regenerate the original tree.

Syntax: `MatrixRep [options];`

Menu equivalent: **Trees >Matrix Representation** ...

Example:

```
matrixrep;
```

Issuing this command will generate a Nexus file with a ”-trees.nex” ending. For example, if your original data file was named ’data.nex’, PAUP* will generate a new Nexus file named ’data-trees.nex’ and save it in the directory along side your original file.

Available Options:

- **File** = ***tree-file-name***
- **Brlens** = **No|Yes**
- **From** = ***starting-tree-number***
- **To** = ***ending-tree-number***
- * **Replace** = **No|Yes**
- * **Append** = **No|Yes**

*Option is nonpersistent

Description of options

Brlens = No|Yes

Use Brlens = Yes to save branch lengths as character weights; a weighted parsimony analysis using these weights will regenerate the original tree and branch lengths.

File = *tree-file-name*

Specifies a name for the tree file. If you do not explicitly specify a file name, a default file name will be used.

Replace = Yes|No

Append = Yes|No

If File = *tree-file-name* option is specified and the *tree-file-name* already exists, you will be prompted for confirmation that the existing file should be replaced. Explicit specification of Replace = Yes suppresses this prompt; the existing file will be quietly overwritten by the new data. Alternatively, you may specify Append = Yes, in which case a new Trees block will be concatenated to the end of an existing file.

From = *starting-tree-number*

If nonzero, *starting-tree-number* specifies the number of the first tree to save.

To = *ending-tree-number*

If nonzero, *ending-tree-number* specifies the number of the last tree to save.

7.33 User-Defined Trees

Tree descriptions follow the ”Newick’s 8:45” standard agreed upon by an informal committee of phylogenetic software developers at the 1986 annual meeting of the Society for the Study of Evolution in Durham, New Hampshire.

User-defined trees are described using a parenthetical notation that defines the shape of the tree. The terminal nodes of the tree correspond to the taxa included in the data matrix, and the internal nodes correspond to hypothetical ancestral taxa. Each pair of parentheses encloses all members of a monophyletic group.

Formally, the system used to describe trees is essentially the same as that for character-state trees (see ”Defining Your Own Character Types,” above). Once again think of the tree diagram as a set of roadways connecting nodes of the tree (see figure below). The itinerary is to visit all of the nodes of the tree (both terminal and internal) in a circuit beginning at the root node, following two simple rules: (1) when you come to an intersection or fork in the road (internal node), always bear to the left, and (2) when you come to a dead end (terminal node), turn around. The path indicated by the arrows shows the sequence in which the nodes would be visited in this example.

To write the tree description, perform the following operations as you make the circuit:

- When you leave a node traveling away from the root toward the leftmost descendant, write a left parenthesis.
- When you leave a node traveling away from the root toward any descendant other than the leftmost descendant, write a comma.
- When you leave the rightmost descendant of an internal node traveling toward the root, write a right parenthesis.

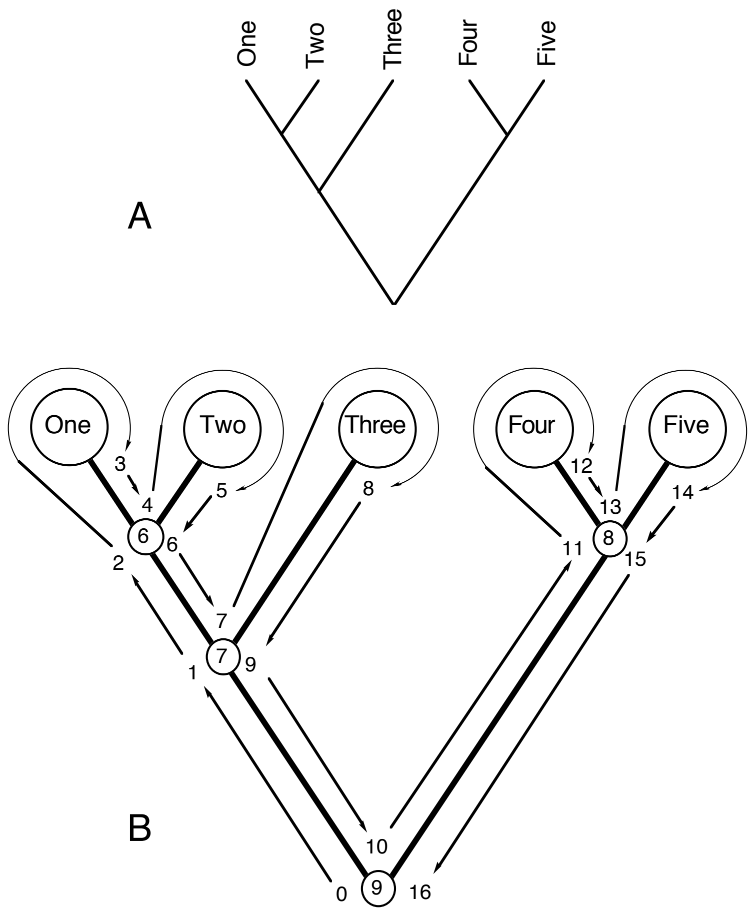


Figure 7.9 *(a) Tree to be input as a user-defined tree; (b) Circuit followed in writing its description.*

- When you visit a terminal node or visit an interior node for the last time, write the node information (if any) for the node.

NOTE: *Usually, the node information will consist of a taxon identifier for terminal nodes and nothing for internal nodes. You are only entering the relative topology of the tree, therefore information about internal node labels and branch lengths is not included. In any case, branch lengths are dependent on a particular data matrix, and are not something that are fixed on a topology. When user-defined trees are read into memory, they can later be output with node labels and branch lengths, although if they are output to a treefile, only the branching topology is preserved. See the sections on "Diagnosing Trees" and "Saving Trees to Files" for descriptions of how trees are described and manipulated.*

Applying these rules to the example of above, the tree description would develop as follows:

```
Number     Description to   Explanation
in         this Point
Sequence

0  (                       leaving for internal node 9's left descendant
1  ((                      leaving for internal node 7's left descendant
2  (((                     leaving for internal node 6's left descendant
3  (((One                  terminal node One visited
4  (((One,                 leaving for internal node 6's next descendant
5  (((One,Two)             terminal node Two visited; leaving internal node
                            6's rightmost descendant
6  (((One,Two)             internal node 6 visited for last time (no node
                            information written)
7  (((One,Two),            leaving for internal node 7's next descendant
8  (((One,Two),Three)      terminal node Three visited; leaving internal
                            node 7's rightmost descendant
9  (((One,Two),Three)      internal node 7 visited for last time (no node
                            information written)
10 (((One,Two),Three),     leaving for internal node 9's next descendant
11 (((One,Two),Three),(    leaving for internal node 8's left descendant
12 (((One,Two),Three),(    terminal node Four visited
    Four
13 (((One,Two),Three),(    leaving for internal node 8's next descendant
    Four,
14 (((One,Two),Three),(    terminal node Five visited; leaving internal
    Four,Five)              node 8's rightmost descendant
15 (((One,Two),Three),(    visiting internal node 8 for last time (no node
    Four,Five))             information written); leaving internal node 9's
                            rightmost descendant
```

```
16 (((One,Two),Three),(   circuit completed (no node information written
    Four,Five))             for root node 9)
```

Note that if the tree is defined as an unrooted tree (i.e., a UTree command), the position of the root implied by the tree description is forgotten once the tree has been successfully stored; other methods (outgroup or Lundberg rooting) must be used to root the tree for output purposes. In this case, the tree can be rooted at any convenient point for input purposes.

All (nondeleted) taxa should be included in the tree description. If a taxon is omitted, it will be joined to the basal node of the tree described by the remaining taxa.

The examples below should clarify some of these points. In particular, observe that trees a and b specify the same unrooted tree, but distinct rooted trees. Also note that the two descriptions shown for tree c are equivalent, since in the second description, omitted taxa "3", "4", and "5" are joined to the basal node of the tree.

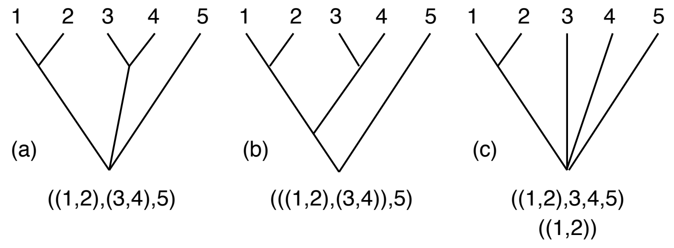


Figure 7.10 *Three trees and their associated tree descriptions.*

Use the UserTree command to input a single user-defined tree. Ordinarily, you should use a Trees block (see "Commands used in the Tree Block") to input one or more user-defined trees. This command merely provides a mechanism for quickly specifying a user-defined tree from the command line, which may be useful in certain situations.

7.34 Random Trees

If you are interested in the tree-length frequency distribution (see "Exhaustive Search" above but the number of taxa is too large to perform an exhaustive search, you can approximate the shape of the distribution by randomly sampling trees under a model in which every possible tree is equally likely. As in the case of an exhaustive search, you may save the tree-length distribution to a file for input into other programs. Use the RandTrees command to randomly sample trees from the set of all possible trees and compute their scores under the current optimality criterion. The results are shown in the form of a frequency distribution of tree scores.

Evaluate Random Trees

Analysis > Evaluate Random Trees...

7.35 Generating random trees

Use the GenerateTrees command to generate a set of random trees or all-possible trees.

If ALL is specified, all possible trees given the total number of terminally labelled taxa are drawn. Depending on the version of PAUP* that you are using, the All modifier may reach the upper limit on the number of trees that can be stored. For example, if only unrooted strictly bifurcating trees are drawn, then a maximum of 8 taxa for 16-bit systems, 12 taxa for 32-bit systems, and 19 taxa for 64-bit system can be included in the analysis. When Random is specified, PAUP* will draw trees according to a Equiprobable or Markovian model (see below).

Example:

The following will evaluate the lengths of 10000 random trees with the seed set to 445667:

Generate Random Trees

- Select **Analysis > Evaluate Random Trees...**
- Type: `1000` in the box after **Evaluate**, then set the **Seed:** value to `445667` and click **OK**

Output consists of a tree-length frequency distribution, which may be saved to a file as in an exhaustive search. For the sample ”cats.nex” data set, the resulting output is as follows:

Output from Generate Random Trees

```
Evaluating 1000 trees sampled equiprobably from the set of
    all possible trees

  Trees are unrooted
  Starting seed = 445667
  Optimality criterion = parsimony
    Character-status summary:
      Of 363 total characters:
        All characters are of type 'unord'
        All characters have equal weight
        212 characters are constant
        55 variable characters are parsimony-uninformative
        Number of parsimony-informative characters = 96
    Gaps are treated as "missing"

  Time used for tree-score evaluation = 0.01 sec (CPU time =
       0.00 sec)

Frequency distribution of lengths of 1000 random trees:

         mean=461.216000 sd=13.796063 g1=-0.597172 g2
             =0.279400
407.000
    /-------------------------------------------------------------------------------

415.400 |# (4)
423.800 |## (7)
432.200 |###### (19)
440.600 |################ (54)
449.000 |######################### (86)
457.400
    |#####################################################
    (178)
465.800
    |######################################################################
     (233)
474.200
    |############################################################################
     (253)
482.600 |###################################### (129)
491.000 |########### (37)
        \-------------------------------------------------------------------------------
```

It is interesting to compare these results with those from the branch-and-bound search where we obtained the number of trees of length less than or equal to 53 steps for the same data set (see "Branch-and-Bound Search" above). Even though there were 49,181 trees of length less than or equal to 53 steps (3 steps longer than the minimum-length trees), the shortest tree found by random sampling of 10,000 trees was 71 steps. Obviously, given that for this 16-taxon data set, there are over 2.134 x 1014 possible (unrooted) trees, random sampling of trees is not an effective way to search for optimal trees.

Available Options:

- **Model** = **Equiprobable|Markovian**
- **NTrees** = ***integer-value***
- **Seed** = ***integer-value***
- **Rooted** = **Yes|No**
- **Nonbinary** = **Yes|No**
- **Constraints** = ***constraint-name***
- *****Converse** = **No|Yes**

*Option is nonpersistent
Descriptions of options

Model = Equiprobable|Markovian

If Random is specified, the Equiprobable model dictates that every possible tree has the same probability. The Markovian model assumes that trees are generated by a random (pure-birth) branching process.

NTrees = *integer-value*

If Random is specified, NTrees gives the number of trees to be generated. The default is 100.

Seed = *integer-value*

If Random is specified, PAUP* references the system clock to obtain a value used to seed a random number generator. An explicit seed may be specified to override the system clock default. (See ”InitSeeds = 0|1” for more details.)

Rooted = { Yes | No }

If Rooted = Yes, rooted trees will be generated, otherwise, the generated trees will be unrooted.

Nonbinary = { Yes | No }

When All is specified, Nonbinary = Yes requests that trees are drawn from the set of all possible binary and nonbinary trees. By default (Nonbinary = No), only binary trees are generated.

If All is specified the following options apply (see ”Options And Subcommands Affecting Multiple Commands” or descriptions of these options).

- Constraints =constraint-name
- Converse = No|Yes

7.36 Pseudorandom Number Generation

A few of the capabilities provided by PAUP* require generation of sequences of ”random” numbers (i.e., random-addition sequence in stepwise addition, sampling of characters in bootstrap analysis, and evaluation of the lengths of random trees). PAUP* uses a linear congruential method: starting with any number xi between 1 and 2,147,483,646 (inclusive), the next number in the sequence is given by x_i+1 = 397,204,094 * x_i mod (231 - 1) (see Fishman and Moore, 1982).

The first number represents the ”seed”. PAUP* always uses ”1” as the initial seed rather than an arbitrary value such as one obtained from the system clock. The main reason for this decision was so that runs from the same data file would always generate identical results on any kind of computer, even if the user neglected to define the initial seed explicitly. The drawback to this approach is that unless the default initial seed is explicitly overridden, exactly the same analysis will be performed in any two runs of the program with identical command sets. For example, conducting a second random-addition-sequence search on a second set of 100 replicates will produce exactly the same set of trees as an initial set of 100 replicates if the program is restarted between the two searches unless you override the default initial seed. Keep this in mind.

8 Statistical Tests

8.1 Goodness-of-Fit Statistics

PAUP* outputs several indices that measure the "fit" of characters to particular trees. Three main parameters are used to define these indices (Kluge and Farris, 1969; Farris, 1989b,a):

s = length (number of steps) required by the character on the tree being evaluated

m = minimum amount of change that the character may show on any conceivable tree

g = maximum possible amount of change that a character could possibly require on any conceivable tree (i.e., the length of the character on a completely unresolved bush).

The **consistency index**, or **CI** (Kluge and Farris, 1969) for a single character, c, equals m/s. Thus, if a particular tree explains the data as well as any tree possibly could, $c = 1$. Unfortunately, the lower bound on c is not 0 but is a function of the distribution of character-states in the data matrix. For example, if two taxa have state 1 but all others have state 0, the maximum possible number of steps on any tree is 2, so that c can be no lower than 0.5. Farris (1989b,a) proposed two new indices, the **retention index (RI)** and the **rescaled consistency index (RC)**. For a single character, the retention index, r, is defined as (g - s)/(g - m). Thus when a character fits the tree as poorly as possible, its retention index will be 0. Note that for uninformative (e.g., autapomorphic) characters, m = g so that r is undefined. Farris (1989b,a) recommends using r as a factor for scaling c between 0 and 1, defining the rescaled consistency index as the product of r and c (= rc).

An "ensemble" (overall) consistency index C (Farris, 1989b) for a suite of characters is calculated as M/S, where M and S are the sums over all characters in the suite of the individual-character m and s values, respectively. The ensemble retention index R is defined analogously to the ensemble consistency index; i.e., (G - S)/(G - M) = $(\sum g - \sum s)/(\sum g - \sum m)$. Archie (1989) independently proposed an equivalent index, calling it the "homoplasy excess ratio maximum" (=HERM). The product of R and C is referred to as the "ensemble rescaled consistency index."

In general, the **homoplasy index (HI)** is equal to 1 - C. But when multistate taxa are treated as "polymorphic," the homoplasy index has a slightly different meaning. In that instance, PAUP* identifies an "ancestral" character state in the multistate taxon from which all observed states in that taxon must be derived (see "Multistate Taxa"). Thus the homoplasy index will not be the same for a character in which multistate taxa are treated as polymorphic (character change allowed within the terminal), and "uncertain" (no character change allowed within the terminal).

All of these measures are useful not only in comparing characters on a single tree, but on multiple trees as well. If several different analyses of a data set produce different trees when different assumptions are made, these indices are a quick way to estimate which characters support which hypotheses of topology. When you use the Pscore command or the **Trees >Tree Scores >Parsimony** menu command, you can output a summary of length and fit measures for trees and for characters. The LenFit command may also be used to calculate these values from the command-line.

8.1.1 Calculating Tree lengths and goodness-of-fit parsimony statistics

Use the Pscore command to request a listing of tree lengths and/or fit measures for one or more trees.

Trees > Tree Scores > Parsimony...

8.2 Assessing Confidence using Bootstrap Analysis

PAUP*'s implementation of **bootstrap** analysis follows Felsenstein (1985). The method involves sampling the original data set with replacement to construct a series of bootstrap replicates of the same size as the original data set. Each of these is analyzed, and the variation among these replicate estimates is taken be an indication of the error involved in making estimates from the original data. In Felsenstein's approach, the taxa are held constant and the characters are sampled with replacement to build a series of new data sets the same size as the original. These are then subject to a search, either heuristic or branch-and-bound. Finally, a majority-rule consensus is constructed for all of the bootstrap trees. If a group appears in X percent of the bootstrap trees, the confidence level associated with that group is taken as X percent. This method gives the investigator the ability to assign statistical confidence to hypotheses of relationship.

There are a number of assumptions underlying the bootstrap, which are discussed in Felsenstein (1985) and in more detail in Sanderson (1989). The utility of the method, like all methods, depends on the validity of these assumptions. We will not go into them in detail here, but perhaps the most important is the ″i.i.d″ assumption identified by Felsenstein (Felsenstein, 1985), which requires that the characters be identically and independently distributed. This is separated into two somewhat less restrictive assumptions by Sanderson (1989), namely that characters are independent, and that the observed character set is a ″representative″ sample of the ″universe of characters.″ It is especially important to remember that if the sample of characters does not accurately reflect the larger underlying distribution of characters, then the bootstrap confidence intervals may be very poorly estimated. Also, the bootstrap cannot overcome systematic biases such as ″long branch effects.″ In any event, it is wise not to take the bootstrap confidence values as absolutes-there are many factors that might lead to over- or underestimates of confidence.

Menu equivalent: **Analysis >Bootstrap/Jackknife** ...

***Example 8.1*:**

In the following example the Bootstrap command is used to execute a heuristic search with random addition sequence on 200 bootstrap replicate data sets. Optimal trees for each replicate are saved to a file named ″boot.tre.″

- Select **Analysis >Bootstrap/Jackknife...**

- After the **Number of replicates:** option, Type: `200`
- Click the **Save trees to file** option
- Chose where you would like to save the file and Type: `boot.tree` in the box labeled **Save treefile as:** and chose **Save**
- Chose **Continue**
- Next, under the **Stepwise-Addition Option**, select the **random** dialog and chose **Search**

The search options for the bootstrap are the same as those for either a heuristic or branch-and-bound search. Because it randomly samples the data matrix with replacement, you must also specify a starting seed for the analysis. If you do not, 1 is used for the first replicate (the number is reset randomly for every subsequent replicate). The following command will begin a branch-and-bound bootstrap search with the initial seed set to 12322, ten replicates, and a confidence level of 50%.

***Example 8.2*:**

- Select **Analysis >Bootstrap/Jackknife...**
- After the **Number of replicates:** option, Type: `10`
- After the **Random number seed:** option, Type: `12322`
- Select the **Branch-and-bound** option under **Type of search**
- Next, click on **Continue**, then select **Search**

If you do not specify a starting seed for subsequent runs, the seed defaults to the next number in the random number sequence initiated during the previous run. You can try different addition sequences and heuristic search options, or you can use a branch-and-bound search. You may specify that compatible trees that are below the confidence level be kept by using the KeepAll option.

***Example 8.3*:**

Command-line

- Type: `bootstrap bseed=1 nreps=10 method=bandb conlevel=50 keepall;`

Be aware, however, that when KeepAll is in effect, some groups may appear in the bootstrap consensus tree that are incompatible with other groups that did not appear. For example, If two groups are different resolutions of a polytomy, they will be incompatible with one another, yet one of them may be included in the consensus and one not, although they appear at similar frequencies. The solution to this is to carefully examine the plot of partition frequencies for other incompatible groupings with similar frequencies-there may be one group found in 32% of the replicate trees that appears in the bootstrap consensus and another conflicting group found in 31% that therefore could not be included.

The presence of unequal character weights adds an additional complication to the use of the bootstrap. PAUP* simply ignored any weights that might be in effect and weighted each character equally for the bootstrap analysis. This option is retained in Version 3.1, and two new options have been added. The first (Wts = Simple) simply assigns each character an equal probability of being sampled, but then uses the weight attached to each character during the following search. The second option (Wts = RepeatCnt) treats each character weight as if it represents the number of times the character was observed. (For example, the weights might represent the number of times a particular pattern of character states was observed, with one column of the data matrix used to represent each pattern). Obviously, these two interpretations have very different implications, and you should decide which one makes the most sense for your analysis. Implied confidence will generally be higher with weights treated as repeat counts, because there will be, on average, more ”characters” supporting each retained group. On the other hand, if the weights are drawn from an especially appropriate biological criterion (e.g., first and second vs. third positions in protein-coding DNA sequences), the ”simple” interpretation is probably more justifiable.

The output of the bootstrap procedure consists of (1) a table showing all partitions (or groups) that were found in the bootstrap replications and their frequencies, and (2) a bootstrap majority-rule consensus tree. The numbers on the branches of the consensus tree indicate the percentage of the bootstrap replications that support the group descending from that branch.

```
Partitions found in one or more trees and frequency of occurrence:


12345678    Freq
----------------
...*..**   91.01
......**   78.66
..******   77.66
...*****   63.40
...*.***   42.83
```

```
..**.***   25.80
.**.**..    8.80
.....***    8.79
....**..    4.34
.**.....    4.07
.*..*...    3.09
..*..*..    2.39
..*.**..    0.78
.**.*...    0.40
..*.*...    0.33
.**..*..    0.30
.*..**..    0.30
..*.****    0.17
..*..***    0.12
```

The frequencies indicate the number of bootstrap replicates in which the particular partition was found. These are called ″partitions″ because the trees are unrooted: either the group represented by periods or the group represented by asterisks would be monophyletic, but not necessarily both (each branch ″partitions″ the taxa into those on one side of the branch and those on the other). Of course, if the trees were intrinsically rooted due to the inclusion of an ancestral taxon or the presence of directed character types, then the partitions would correspond directly to monophyletic groups on the tree. If *N* multiple equally parsimonious trees are found in a given replicate, each group found on each of these trees contributes 1/*N* to the total group count. This explains the presence of fractional values in the partition-frequency table.

The bootstrap consensus tree shows the relative partition frequencies (expressed as a percentage) corresponding to each branch. Note that percentages on the consensus tree are rounded.

```
Bootstrap 50\% majority-rule consensus tree

?<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
,àë<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
>            ?<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
>            >                          ?<<<<<<<<<<<<<<<
?<<<<<78<<<<<?            ?<<<<<91<<<<<          <<<<<<
             >            >             ?<<<<<79<<<<<<<<
             ?<<<<<63<<<<<<
                          ?<<<<<<<<<<<<<<<<<<<<<<<<<<<<
```

The consensus tree that is ouput for the bootstrap will in general be longer than the minimal tree for the full data set, although this is a function of consensus trees in general and is not unique to the bootstrap. This is because in PAUP*, polytomies are treated as representing simultaneous multiple splitting (”hard polytomies”) (Maddison, 1989) rather than as indicating uncertain resolution (”soft polytomies”).

8.3 Assessing Confidence using a Jackknife Analysis

Like the bootstrap, the jackknife (Farris et al., 1996) is a statistical technique used in resampling statistics. This method consists of forming new samples by omitting, in turn, one or more of the observations (observed characters) of the original data. For each of the sub-samples generated, the estimator under study (tree topology) can be re-estimated, and the probability distribution thus obtained will allow one to draw conclusions about the estimator’s sensitivity to individual observations. Ideally, the rest of the phylogeny should remain unchanged from sub-sample to sub-sample.

Analysis >Boostrap/Jackknife ...

8.4 Permutation tests

8.4.1 Archie-Faith-Cranston randomization test

Analysis > Permutation Tests...

8.5 Partition homogeneity test

Use the HomPart command to perform test for homogeneity of partitioned data sets. This test was described as the incongruence-length difference test by Farris et al. (1995).

Analysis > Partition Homogeneity Test...

***Example 8.4*:**

The following example of the HomPart command uses the character partition definition named ”genes”, specifies 1000 randomizations using the random number seed 123, and uses a branch and bound search to obtain the sum of tree lengths for each partition.

Command-line

- Type: `charpartition genes=cytb:1-200, 12S:201-.;`
- Type: `hompart partition=genes nreps=1000 seed=123 search=bandb;`
- Select **Analysis > Partition Homogeneity Test...**
- Select the **Character Partition:** ”genes” from the pull-down menu, select appropriate options in other dialog boxes and click **Continue**.
- Select appropriate Branch-and-bound settings and click **Search**

8.6 Topology Tests

Different tree topologies represent competing evolutionary hypothesis. A simple and objective way to discriminate among trees is to compare the scores of topologies under one of PAUP*'s criterion-based method. Generally speaking, if the difference between two tree scores is large then you might be inclined to accept the tree with the better score as a better representation of the true evolutionary history. The relative difference between tree scores by itself, however, is not entirely satisfying. Ultimately, we would like to say with some degree of confidence that certain trees explain the data better than others. Table 1 summarizes the methods for comparing tree topologies that are implemented in PAUP* and gives the corresponding command options used to execute them from the command-line. Further details regarding these methods are given in the following paragraphs.

Test	Null Distribution	Criteria Supported	Command
KH-test	Normal	Parsimony	`pscores <tree-list>/khtest=yes;`
		Likelihood	`lscores <tree-list>/khtest=normal;`
	Bootstrap (Fully Optimized)	Likelihood	`lscores <tree-list>/khtest=fullopt;`
	Bootstrap (RELL)	Likelihood	`lscores <tree-list>/khtest=rell;`
SH-test	Bootstrap (Fully Optimized)	Likelihood	`lscores <tree-list>/shtest=fullopt;`
	Bootstrap (RELL)	Likelihood	`lscores <tree-list>/shtest=rell;`
Templeton	Normal Approximation	Parsimony	`pscores <tree-list>/nonparamtest=yes;`
Winning Sites	Binomial	Parsimony	`pscores <tree-list>/nonparamtest=yes;`

8.6.1 The Kishino-Hasegawa test

Hasegawa and Kishino (1989) and Kishino and Hasegawa (1989) proposed methods for estimating the variance of the difference between the scores of competing tree topologies. In PAUP*, these methods fall under the general heading of the Kishino-Hasegawa test (hereafter referred to as the KH-test). The motivation behind the development of this class of methods is to test whether sampling error could possibly explain the difference between tree scores. For example, if the difference between two trees scores (D) was merely the result of sampling error then on average the expected value of D would equal zero. We can state the hypotheses of the KH-test more succinctly as follows:

- H0: E[D] = 0
- HA: E[D] $\neq$ 0

The Null hypothesis, as stated above, is only true when the trees being compared are selected before knowing which of the two trees fit the data better. For example, the expected value of *D* must be greater than zero for a comparison between the optimal tree for the data and any other tree.

For this reason the KH-test is strictly only valid when the trees being compared have been selected *a priori* (Swofford et al., 1996; Goldman et al., 2000).

PAUP* allows the user to select among several methods for generating a null distribution.

8.6.2 Full optimization

To create a confidence interval for D Hasegawa and Kishino (1989) proposed using a nonparametric bootstrap resampling technique (Efron, 1979; Felsenstein, 1985). The basic bootstrap technique used in this test is no different from that used to generate confidence intervals on individual phylogenies (see section "Assessing Confidence using Bootstrap Analysis"). In this case, however, we are only interested in calculating the difference between the scores of competing tree topologies ($\boldsymbol{D_i}$) for each replicate data set. To ensure that the test distribution is consistent with the null hypothesis (i.e., mean of $\boldsymbol{D_i}$ is equal to zero) a centering procedure is also employed. This procedure subtracts the mean of $\boldsymbol{D_i}$ over all replicates from each replicate $\boldsymbol{D_i}$.

8.6.3 RELL

8.6.4 Normal

The test assumes that the character sites used to evaluate the tree topologies are independently and identically distributed (i.i.d.). If the trees being compared were selected before knowing which of the two trees fit the data best then the null hypothesis is that the difference between the tree topologies is equal to zero. If however, you are comparing the optimal tree for the data with another tree topology then the null is no longer zero and a one-tailed test should be used. In either case the test statistic is the difference between the tree scores.

8.6.5 Test distributions

Test is two or one sided
Bootstrap with full optimization

Bootstrap with RELL optimization
SH-test
Trees selected a posteriori
Likelihood only
H0 = Average difference between trees is equal zero
H1 = Average difference between trees is not equal to zero
Templeton's Test
Winnning Sites
Test Number of trees compared Null distribution choice of trees to test

Khtest 2 a priori
SHtest 3> a posteriori
Winning Sites
Templetons

8.6.6 The Shimodaira-Hasegawa Test

8.6.7 Shimodaira's Approximately Unbiases (AU) Test

8.7 Base frequency composition

8.7.1 Examining base frequency composition

Use the BaseFreqs command to show base frequencies for each taxon. In addition, this command gives the expected base frequencies for each taxon and the Chi-square test of homogeneity of base frequencies across taxa.

Syntax: : `BaseFreqs;`

Menu equivalent:

Data > Base/AA Frequencies

9 Parsimony Settings

9.1 Character-State Optimization

The reconstruction of character states at internal (ancestral) nodes on a given tree is called character-state optimization or character mapping. Character optimization does *not* come into play at any time during the search for optimal trees; *only* when character reconstructions are requested. Under the maximum parsimony criterion, the goal is to assign character states so as to minimize the total number of change required by a particular character on a given tree. The set of such assignments at internal nodes is called a most-parsimonious reconstruction (MPR) or simply an ”optimal reconstruction” (Swofford and Maddison, 1987, 1992). Characters do not have to be polarized in order for these reconstructions to be made.

The optimal reconstruction for a given character is a function of tree topology, but also of any other assumptions about character order and/or irreversibility that have been made. Specifically, this involves the ”cost” of changing from one character state to another. This is a function of the character type, whether one of the standard types or user-defined. The character optimization algorithm operates by making two passes over a tree. The first pass proceeds from the tips toward the root. At each internal node, the algorithm determines which character states are potentially allowed, and calculates the minimum possible lengths of the clade above that node for each allowed state.

The second stage proceeds from the root to the tips, and determines a set of optimally reconstructed internal states by using the information from the first pass and information from the second pass below that node. The optimal state minimizes the sum of two quantities: the length of the clade above that node given each allowed state and the cost of transforming to each allowed state given the state assigned to the ancestor of that node.

In most cases there will only be one optimal assignment at a node, but it is possible that more than one optimum may exist. In that case, a particular reconstruction may be favored on the basis of additional criteria. The most common ancillary criteria are accelerated transformation (ACCTRAN) and delayed transformation (DELTRAN) (Swofford and Maddison, 1987; Maddison and Maddison, 1992; Swofford and Maddison, 1992). Basically, these assign states at internal nodes in order to delay (DELTRAN) or accelerate (ACCTRAN) character transformation within the tree. By ”pushing” transformations up or down the tree, these optimization methods lead to very different hypotheses of character change. Delaying change will lead to a preference for two origins of a character state (parallelisms), while accelerating change will lead to a preference for a single origin followed by reversal.

Both hypotheses will always involve the same number of steps on the tree. The only change will be where those steps are located. The result will be that the character change associated with a particular node will vary with the choice of optimization method. For that reason, you have to carefully examine the output that PAUP* gives you (see ”Character Diagnostics”).

In the figure below, assume that the ancestral state is 0, and that taxa *A*,*B*,*C*, and *D* have the state assigned them for that character. There are two ways that this character can be optimized: either the occurrence of state 1 in *B* and *C* is a reversal (reconstruction a), in which case 1 arose at the level of (*ABC*) and was lost in *A*; or the presence of 1 in *B* and *C* is due to parallelism (reconstruction b), or independent gain. In either case, this character requires two steps when it is optimized, so the length of the tree does not change.

This is especially important if an analysis is explicitly examining the levels of parallelism/reversal in a data set. In that instance, there are several steps that one might take. Most obviously, use both optimization methods, and see how the characters fall out-it may be that there is no difference. Another strategy is to choose the method which best favors

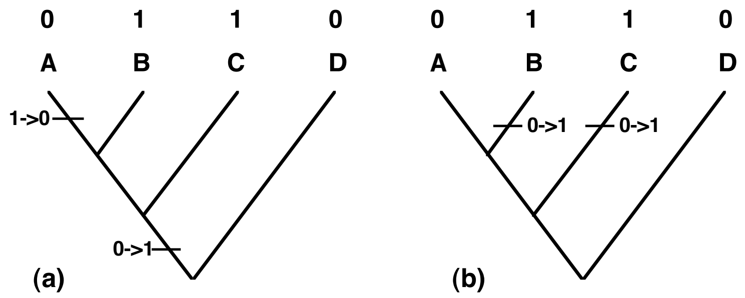


Figure 9.1 *Character optimization using ACCTRAN and DELTRAN. (a) ACCTRAN optimization of character with distribution shown. (b) DELTRAN optimization of character. Both require two steps.*

the null hypothesis. For example, if you are studying adaptation and want to discover instances of parallel derivations of a trait, choose ACCTRAN, which favors reversals. If a pattern of repeated parallelisms *still* appears in spite of a bias against them, the argument for adaptation is that much stronger.

A third option for character-state optimization is available: MINF. Under MINF optimization, character states are optimized so that the f-value of Farris (1972) is minimized. Two constraints are enforced: the states assigned to an HTU must be present in at least one OTU, and the tree length must be minimal (it is possible to further reduce the f-value by increasing the length of the tree). The effect of minimizing the f-value is that length is transferred from interior branches towards terminal branches whenever possible, minimizing the risk that groups will be arbitrarily resolved internally. This option will, in many instances, yield the same reconstruction as DELTRAN. MINF is not available for rooted trees.

9.2 Setting parsimony options

Use the PSET command to set options for parsimony analysis.

Syntax: `: PSet` [*options*];

Menu equivalent: **Analysis >Parsimony Settings** ...

Available Options:

- **Collapse = No|MaxBrlen|MinBrlen|AmbEqual**
- **MSTaxa = Uncertain|Polymorph|Variable**
- **Opt = AccTran|DelTran|MinF**
- **StepMatrix = ObsOnly|AllStates|ThreePlus1**
- **IncludeAnc Yes|No**
- **AncStates = ancstates-name**
- **GK = integer-value**
- **Goloboff = Yes|No**
- **GUninf = Exclude|Include**
- **GPeeWee = Yes|No**
- **MinForFit = MinLength|Range**
- **GapMode = Missing|NewState**

Description of options

Collapse = No|MaxBrlen|MinBrlen|AmbEqual

Use COLLAPSE to specify the criterion for collapsing branch lengths. MAXBRLEN is the default method used in PAUP* and the method used in PAUP* (and Hennig86). MINBRLN collapses a branch if it is possible for it to have zero length. AMBEQUAL collapses a branch if the MPRSET of the two incident nodes are identical. These latter two methods were introduced in Goloboff's NONA program. We do not recommend their use, but they are available for those who wish to use them.

MSTaxa = Uncertain|Polymorph|Variable

If one or more taxa are coded as having multiple states use MSTAXA to specify how those characters are treated. If MSTAXA = VARIABLE the punctuation used to enclose multistate characters is respected. More specifically, characters enclosed by "" are treated as variable and characters enclosed by "()" are treated as polymorphic.

MinForFit = MinLength|Range

If MSTAXA is specified, then use MINFORFIT to specify the ”minimum” values used for calculating CI, RI, and RC indices. You may specify minimum-possible single-character lengths (MINLENGTH) or character ”ranges” (RANGE) be used.

StepMatrix = ObsOnly|AllStates|ThreePlus1

By default PAUP* allows assignment of states not observed in terminal taxa to internal nodes but only those states that can be identified as potential shortcuts by the ”3+1” test (STEPMATRIX = THREEPLUS1). If STEPMATRIX = ALLSTATES, then any possible character state may be assigned to an internal node. If STEPMATRIX = OBSONLY, then internal-node state assignments are limited to states observed in the terminal data.

IncludeAnc = Yes|No

Use INCLUDEANC to include a hypothetical taxon possessing putative ancestral states for all characters in the searches. This taxon specifies the root of the tree.

Goloboff = Yes|No

Use GOLOBOFF to specify the Goloboff-fit criterion.

GK = *integer-value*

Use GK to specify the concavity parameter in Goloboff’s implied weights method. Note that this value corresponds to the definition of K given by Goloboff (1993) and not the CONC parameter in Goloboff’s Pee-Wee program; which is equal to $K + 1$. The default value for this option is 2.

Guninf = Exclude|Include

If GUNINF = INCLUDE, then fits for ”uninformative” characters are included.

GPeeWee = Yes|No

Although it is *not recommended*, you may specify GPEEWEE to emulate results obtained by the program Pee -Wee written by P. Goloboff. Calculations are performed using integer arithmetic with weights ranging from 0 to 100, and fractional values are truncated rather than rounded.

GapMode = Missing|NewState

If GAPMODE = MISSING, gap characters in sequence data are treated as ”missing.” If GAPMODE = NEWSTATE, gap characters are treated as a fifth base or 21st amino acid.

See ”Options And Subcommands Affecting Multiple Commands” for a description of the following options:

AncStates = *ancstates-name*

Opt = AccTran|DelTran|MinF

10 Likelihoods Settings

.

10.1 Using maximum likelihood to infer phylogenies

In addition to parsimony, PAUP* also use the maximum-likelihood criterion to evaluate or infer evolutionary trees. Under maximum-likelihood, an explicit model of nucleotide, protein, or morphological substitution, along with the observed data, is used to evaluate trees. In short, the maximum-likelihood method selects the hypothesis (H) that maximizes the probability of obtaining the observed data (D).

$$L_D \propto Pr(D|H) \tag{10.1}$$

Where $Pr(D|H)$ is equal to the probability (conditional probability) of observing the **data** (D) given **hypothesis** (H). In the context of phylogenetic inference, the hypothesis is a tree topology (i.e., the branching order of the sequences as well as the branch lengths) and the observed data are DNA, protein, or morphological data. Each competing hypotheses (i.e, trees) are evaluated under the proposed model of character evolution. The tree that makes the observed character the most probably evolutionary hypothesis is the **maximum likelihood estimate** of the phylogeny. Of course, as the number of taxa, or sequences in the data matrix grows, so too does the number of possible hypotheses that need to be evaluated (see Section 2.3.1).

Under the assumption that the observed characters are evolving independently, it is possible to calculate the likelihood of each observed site (**site likelihood**) separately, the combine the likelihoods into a total value (**tree score**) at the end.

$$L_{total} = L_1 L_2 ... L_N = \prod_{j=1}^{N} L_j \tag{10.2}$$

where N equals the number of aligned sites. Because the probability of any single observation is an extremely small number (much to small to be represented using standard floating-point representation on modern computers), it is convention to evaluate the log of the likelihood, that way the probabilities are calculated as the sum of all the single-site likelihood values, as such:

$$lnL_{total} = lnL_1 + lnL_2 + ... + lnL_N = \sum_{j=1}^{N} L_j \tag{10.3}$$

In PAUP* the tree scores are given as the negative log-likelihood ($-lnL$).

10.2 Models of DNA sequence evolution

In order to calculate the likelihood of a tree, we need a model that describes the probability of transition of characters from one state to another. This model may be fully specified, or alternately, can contain any number of parameters that are estimated from the data. These models for DNA substitution in PAUP* includes a description of the substitution process (called the **substitution model**) and are the only substitution models described in this section. All the models in PAUP* have the property of being **time reversible**, that is, it can be assumed to be memoryless. Meaning that any particular nucleotide at a particular site has a certain probability of changing at any given instance to another nucleotide, regardless of what nucleotide was present at any given instant in the past. For example, if a sequencce position has base A at some time t_0, the probability that it will be base T at some later time t_1 depends only on the fact that it has base A at t_0; knowing that it had state C at some time prior to t_0 would be irrelevant to the probability. Such processes are referred to as **Markov processes**.

The mathematical expression of a substitution model is a table of rates (substitutions per site unit of evolutionary distance) at which each nucleotide by each alternative nucleotide. For DNA sequences, these rates can be expressed as a 4 x 4 instantaneous rate matrix, **Q**, in which the element Q_{ij} represents the rate of change from base i to j during some infinitesimal time period dt. The most general [often called general time-reversible, or GTR (Lanave et al., 1984; Tavare, 1986; Barry and Harrington, 1987; Rodriguez et al., 1990) OTHER CITATIONS] is the following matrix:

$$\mathbf{Q} = \begin{pmatrix} -\mu(a\pi_C + b\pi_G + c\pi_T) & \mu a\pi_C & \mu b\pi_G & \mu c\pi_T \\ \mu a\pi_A & -\mu(g\pi_A + d\pi_G + e\pi_T) & \mu d\pi_G & \mu e\pi_T \\ \mu b\pi_A & \mu d\pi_C & -\mu(h\pi_A + j\pi_C + f\pi_T) & \mu f\pi_T \\ \mu c\pi_A & \mu e\pi_C & \mu f\pi_G & -\mu i\pi_A + k\pi_C + l\pi_G) \end{pmatrix}$$

where the rows (and columns) correspond to the nucleotides A, C, G, and T, respectively. The factor μ represents the **mean instantaneous substitution rate**. This mean rate parameter is weighted by a relative rate parameter $a, b, c, ..., l$, which corresponds to each possible substitutions from one nucleotide to another nucleotide. The product of a relative rate parameter and the mean instantaneous substatution rate corresponds to a **rate parameter**. The remaining parameters, π_A, π_C, π_G and π_T, are **frequency parameters** that correspond to the frequencies of bases A, C, G, and T, respectively (Yang, 1994a). We assume that these frequencies remain constant over time (i.e., they are in equilibrium) and that the rate of change *to* each base is proportional to the equilibrium frequencies but independent of the identity of the starting base. The diagonal elements of **Q** are always chosen so that the elements in the corresponding row sum to zero.

Almost all of the DNA substitution models proposed to date are special cases of the **Q** matrix shown above. Most of the remaining models used for phylogenetic inference using maximum likelihood or estimation of pairwise distances can be obtained by restricting the parameters of the GTR model (Figure 9.1). For example, if the substitution types are divided into transversions, transitions between purines and transitions between pyrimidines, the model becomes the TrN (Tamura and Nei,

1993) by requiring that $a = c = d = f$. Likewise, we can obtain the three-substitution model of Kimura (Kimura, 1981, K3ST) by requiring that all bases occur at equal frequencies (π_A, π_C, π_G and $\pi_T = 0.25$) and dividing the substitution types into transitions ($b = e$), A $\leftrightarrow$ T or C $\leftrightarrow$ G transversions ($c = d$), and A $\leftrightarrow$ C or G $\leftrightarrow$ T transversions ($a = f$).

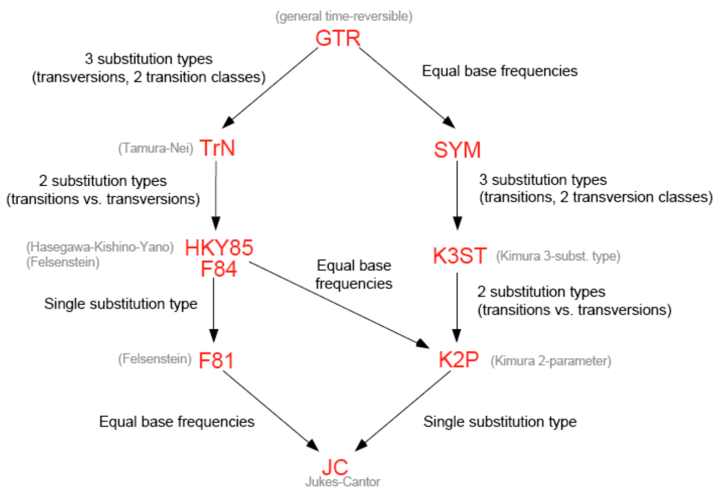

(general time-reversible)
GTR
3 substitution types (transversions, 2 transition classes)
Equal base frequencies
(Tamura-Nei) TrN
SYM
2 substitution types (transitions vs. transversions)
3 substitution types (transitions, 2 transversion classes)
(Hasegawa-Kishino-Yano) HKY85
(Felsenstein) F84
Equal base frequencies
K3ST (Kimura 3-subst. type)
2 substitution types (transitions vs. transversions)
Single substitution type
(Felsenstein) F81
K2P (Kimura 2-parameter)
Equal base frequencies
Single substitution type
JC
Jukes-Cantor

Figure 10.1 *Relationships between special cases of the general time-reversible family of substitution models. Arrows represent restrictions that convert a more general model to a more specific one. Model abbreviations: F81, model of Felsenstein (1981b), (equivalent to the "equal input" model of Tajima and Nei (1982); F84, model used in versions 2.6 and later of PHYLIP (Felsenstein, 1991; Kishino and Hasegawa, 1989); GTR.*

Further restrictions on the parameters in the **Q** matrix lead to more familiar models. For example, if we assume equal base frequencies (π_A, π_C, π_G and $\pi_T = 0.25$) and that all substitutions occur at the same rate ($a = b = c = d = e = f = 1$), the model reduces to that of Jukes and Cantor (1969, JC69)

$$\mathbf{Q} = \begin{pmatrix} -\frac{3}{4}\mu & \frac{1}{4}\mu & \frac{1}{4}\mu & \frac{1}{4}\mu \\ \frac{1}{4}\mu & -\frac{3}{4}\mu & \frac{1}{4}\mu & \frac{1}{4}\mu \\ \frac{1}{4}\mu & \frac{1}{4}\mu & -\frac{3}{4}\mu & \frac{1}{4}\mu \\ \frac{1}{4}\mu & \frac{1}{4}\mu & \frac{1}{4}\mu & -\frac{3}{4}\mu \end{pmatrix}$$

The base frequency and substitution rate are usually combined into a single parameter $\alpha = \mu/4$, leading to the simpler matrix:

$$\mathbf{Q} = \begin{pmatrix} -3\alpha & \alpha & \alpha & \alpha \\ \alpha & -3\alpha & \alpha & \alpha \\ \alpha & \alpha & -3\alpha & \alpha \\ \alpha & \alpha & \alpha & -3\alpha \end{pmatrix}$$

Zharkikh (1994) described a model (SYM) that is equivalent to the GTR model except that it assumes equal base frequencies. Any other restriction of the relative rates from the general time-reversible model (e.g., $a = c$, $e = f$) is possible. With 6 substitution classes, there are a possible 203 model combinations; all such models are also time-revesible.

10.3 Models of DNA sequence evolution

10.4 Calculating change probabilities

The instantaneous rate matrix **Q** specifies the rate of change between pairs of nucleotides or amino acids per instant of time dt, but in order to calculate likelihoods we need the **probabilities** of change from any state to any other along a branch of length t. The substitution probibility matrix can be calculated as

$$P(t) = e^{Qt} \tag{10.4}$$

(Hasegawa et al., 1985; Yang, 1994a). The exponential can then be evaluated by decomposing the instantaneous rate matrix **Q** into its eigenvalues and eigenvectors (we omit the details here, but the reader should see the ”Likelihood” chapter in the ”Background Information” section of the book).

10.5 The relationship between substitution rate and time

For all of these models, the probability of a change from state i to state j dependes on the interaction of the duration of time t and the substitution rate μ only through their product μt (Felsenstein, 1981a). Thus, a branch could be "long" either because it represents a long period of evolutionary time or because there is a high substitution rate. In general, it is impossible to tease these two components apart unless one assumes a strict molecular clock. Consequently, the mean substitution rate μ is usually set to 1 and the relative rate parameters $a, b, ..., f$ are scaled so that the average rate of substitution at equilibrium is 1 (Yang, 1994a). The length of a branch then represents the expected number of substitutions per site along that branch, with no implication as to the actual amount of evolutionary time (absolute time) it actually represents.

These models allow for the expected number of substitutions to be different for each branch of a tree. As noted, one consequence of this property is that the likelihood of a tree can be calculated independent of the position of the root.

10.6 Accommodating rate heterogeneity across sites

All the maximum likelihood models discussed so far assume that every site evolves at the same rate. Violation of this assumption can have devastating effects when estimating a phylogeny. If there is strong variation in rates across sites in the data, sites that might be resistant to change (e.g., like sites under strong selective constraints) can hide the actual amount of change that has occurred at more rapidly evolving sites. This will cause maximum likelihood to underestimate the number of multiple substitutions; the longer the branch length, the greater the underestimation. Among site rate variation is a well known characteristic of DNA sequence data that can potentially confound all methods of tree building if not taken into consideration in models of molecular evolution (Kuhner and Felsenstein, 1994; Sullivan and Swofford, 2001) and cause the underestimation of evolutionary distances (Yang, 1996).

Rate heterogeneity can be incorporated into our likelihood analyses by an additional relative rate component, r, into our substitution probability formulas. For example, for the JC model we let:

$$P_{ij}(t,r) = \begin{cases} \frac{1}{4} + \frac{3}{4}e^{\mu rt} & i = j \\ \frac{1}{4} - \frac{1}{4}e^{\mu rt} & i \neq j \end{cases}$$

If the relative rates r are scaled so that the mean substitution rate remains 1, branch lengths will still reflect the number of substitutions per site. In the simplest case, we can simply assign a rate r_j to each site j. Typically, the basis for this assignment of rates to sites would be some *a priori* classification of sites into a functional categories (like codon position) and assignment of relative rates to the categories.

The simplest model based on a discrete distribution is an "invariable-sites" model. This model assumes that some fraction of the sites in an alignment is incapable of changing (perhaps do to strong functional constraint), but the remaining sites are free to vary, but at the same rate (Hasegawa et al., 1985; Churchill et al., 1992; Reeves, 1992; Sidow et al., 1992). In this case, when $r = 0$, $P_{ij}(t,r) = 1$ and $P_{ij}(t,r) = 0$ for all $i \neq j$. The proportion of invariable sites must either be estimated separately or treated as a parameter that is optimized for each tree. There is no reason in principle to restrict the rate of one of the categories to 0 (no change, or to limit the number of categories to 2, but estimation of the proportions of sites within each category and the relative rates among categories becomes much more complicated.

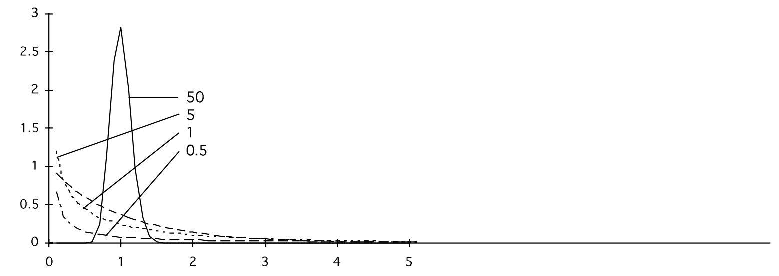


Figure 10.2 *The gamma distribution, with scale equal to the inverse of the shape, for four different values of the shape parameter (α). Note that for high values of the shape parameter, the distribution becomes a sharp peak centered at 1.0, corresponding to near homogeneity of rates across sites, whereas at low values the distribution becomes quite skewed.*

The most commonly used continuous distribution for modeling among site rate variation is the gamma (Γ) distribution (Yang, 1993). The Γ distribution has two parameters, a shape parameter *alpha* (α) and a scale parameter *beta* (β). By setting β to $1/\alpha$, a distribution with mean = 1 is obtained. A wide variety of rate distributions can be obtained by varying the value of α (Figure 9.2).

The shape parameter α is equal to the inverse of the squared coefficient of variation of the substitution rate, so that as α increases, the distribution converges to an equal rates model. Obtaining likelihoods by integrating over Γ (or any continuous distribution) is usually very computational intensive (Yang, 1993). Yang (1994b) proposed an alternative approach in which the Γ distribution is divided into several categories by finding boundries in the distribution such that each category has equal probability. The mean (or median) of each category is then used to represent all the rates within that category. Yang (1994b) found that this ″discrete gamma″ model can provide a good approximation with as few as four rate categories. The discrete Γ distribution only adds one additional parameter to the model (the shape parameter α), no matter how many rate categories are defined. This discrete Γ approach is how PAUP* models among-site rate variation calculates likelihoods of trees.

In some cases, a mixture of rate heterogeneity models may be appropriate. For example, it might be advantageous to model rate heterogeneity with an ″invariant + gamma″ model (I + Γ). In such a case, some fraction of the sites, θ, are invariable, while the remaining sites are distributed according to the Γ distribution with shape parameter α.

10.7 Selecting an appropriate model of nucleotide substitution

Selecting an appropriate model of nucleotide substitution is an important step in a likelihood analysis but is beyond the scope of the manual. There is a vast amount of literature devoted to model selection as it pertains to phylogenetics and the reader is encouraged to see Swofford et al. (1996) for a discussion of model selection under the maximum likelihood criterion. The authors also wish to make clear that there are many ways to select an appropriate of DNA substitution for a set of aligned sequences. Most of these methods attempt to select a model that optimizes the tradeoff between fit to the data and model complexity [e.g., likelihood ratio test (Cox and Hinkley, 1974); Akaike information criterion (Akaike, 1974)]. There are programs [e.g., ModelTest (Posada and Crandall, 1998)] that fully automate the selection of models based on output from PAUP*. However, the model selection procedure can be fully interactive. This approach may be preferable for two reasons: 1) by walking step-by-step through the model selection process, the investigator will because more aware of some of the options available in PAUP* and 2) by selecting a model interactively it is often possible to learn about specific aspects of sequence evolution that might otherwise go unnoticed.

For more general information about maximum likelihood and some of the models available in PAUP*, see the ”Likelihood” chapter in the ”Background Information” section of the manual. In addition, the reader should see Swofford et al. (1996), Lewis (1998), Felsenstein’s book for good treatments of models of sequence evolution and the calculation of likelihood values on evolutionary trees.

10.8 Setting a model for maximum likelihood calculations

As mentioned above, under maximum likelihood, an explicit model of nucleotide substitution is used to evaluate trees. Therefore, the user needs to set the parameters of a given model before evaluating or searching for trees. Use the Lset command to set options for maximum likelihood analysis.

Analysis >Likelihood Settings...

Example 1:

The following example pools transitions into a common rate class and defines two separate rate classes for transversions. Rate classes will be estimated using maximum likelihood.

Menu equivalent

Selecting a likelihood model in PAUP*:

- Select **Analysis >Likelihood Settings...**

- Select the **General time-reversible (”6ST”)** option under the **Substitution Model** pull down menu. Then check the **Estimate** option for the **Rate matrix:**.

- Next, click on the **Specify submodel...** button. You will be presented a interface where you can set the substitution classes. Set **A** ↔ **C**, **A** ↔ **T**, **C** ↔ **G**, and **G** ↔ **T** to **a**; **A** ↔**G** to **b**; and **C** ↔ **T** to **c**

- Click **OK**, then click **OK** again

Example 2:

The following example uses a discrete gamma distribution with four rate categories to estimate across-site rate heterogeneity. The shape parameter of the gamma distribution is estimated using maximum likelihood.

Menu equivalent

- Select **Analysis >Likelihood Settings...**
- Under the **Maximum likelihood options:** pull-down menu, select **Among-site Rate Variation**
- click on the **Gamma distribution** option in the **Variable sites** box. Select to **Estimate** the **Shape parameter**. The default **Number of rate categories** in PAUP* is 4. Leave this unchanged.
- Click **OK**

Example 3:

The following example estimates a single rate for each subset of the data as defined in the Charpartition command. See "CharPartition" on for more detail regarding this command.

Command-line

- Type: `charpartition genes=g1:1-300, g2:301-600, g3:601-700;`

Example 4:

The following example defines a specific rate for each subset of the data as defined by the Rateset command. See "RateSet" for more dtails regarding this command.

Command-line

- Type: `rateset genes=1:1-300, 2:301-600, 3:601-700;`
- Type: `lset rates=sitespec siterates=rateset:genes;`

10.9 Calculating the likelihood of a trees in memory

PAUP* can also evaluate the likelihood and model parameters of a tree, or trees, in memory. This allows the user to calculate likelihood scores and parameter of other tree topologies found under other optimality criteria (e.g. parsimony or distance) or from topologies representing other evolutionary hypotheses (e.g, trees from constraint searches). PAUP* also can evaluate if these different topologies are significantly different from one another using several statistical tests (see below and the ”Statistical Methods” section of this chapter). Once again, it is necessary to chose and set a model of character evolution before evaluating the trees. This can be done by using the Lset command (see above) or while calculating the likelihood of trees in memory (see the ”Lset” command for additional details).

Use the Lscores command to calculate the likelihoods of trees in memory.

Trees >Tree Scores >Likelihood ...

Example:

The following example of the Lscores command returns the emperical base frequencies, the estimated transition/transversion ratio, and the negative log likelihood of all trees in memory.

Menu equivalent

- Select **Trees >Tree Scores >Likelihood...**
- Make sure that **All** trees are selected and then click bf Likelihood Settings
- **NOTE:** the options under **Likelihood Settings** are identical to the options under the LSET command
- under the **Ti rate $\neq$ tv rate("2 ST")** box, click **Ti/tv ratio:** to **Estimate**
- then under the **Base Frequencies** submenu under **Maximum likelihood options:**, select **Use empirical frequencies**

10.10 Displaying the Gamma distribution

Use the GammaPlot command to show a gamma- distribution plot.

Syntax: : `GammaPlot` [*options*];

Menu equivalent: **Options >Gamma Distribution** ...

Available Options:

- **Shape** = ***real-value*|Estimate|Previous**
- **NCat** = ***integer-value***

- **RepRate = Mean|Median**

Description of options

Shape = *real-value*|Previous

If Shape = Previous the plot of the gamma distribution is drawn using the most recently estimated shape parameter (see the ”LSet” command) . To obtain a plot of the gamma distribution with a user-defined shape parameter specify Shape = *real-value*. PAUP* uses the default value of 0.5 for this option.

NCat = *integer-value*

Use NCat to specify the number of categories into which the continuous gamma distribution is divided in order to obtain a discrete approximation. The default is 4.

RepRate = Mean | Median

Use RepRate to specify whether the mean or the median is used represent the rates within each category of the gamma distribution.

10.11 Checking likelihood surface for multiple peaks

Use the SurfCheck command to check likelihood surface for multiple peaks.

Syntax: : `SurfCheck` [*options*];

Available Options:

- **NReps = *integer-value***
- **Seed = *integer-value***
- **TreeNum = *integer-value***
- **BrLens = Yes|No**

Description of options

NReps = *integer-value*

Use NReps to specify the number of passes ...

Seed = *integer-value*

PAUP* references the system clock to initialize the seed used to generate pseudorandom numbers for sampling ”random” branch lengths. An explicit seed may be specified to override the system clock default. (See ”InitSeeds = 0|1” for more details.)

TreeNum = *integer-value*

Use TreeNum to specify the tree for which the likelihood surface is checked.

BrLens = Yes | No

If BrLens = Yes, then user-supplied branch lengths are used.

11 Distance Settings

11.1 Introduction

In addition to parsimony and maximum likelihood, PAUP* also searches for and evaluates trees under the distance criterion (minimum evolution or least squares objective function). However, unlike parsimony and likelihood, distance methods do not rely on the discrete character data, but rather a transformation of those characters into a distance matrix. The way in which the data is transformed is dependent on the data at hand, as well as, assumptions made by the user. Distance methods are thus ”model based,” like maximum-likelihood methods. PAUP* provides a variety of ”corrected” distances for DNA sequence data, but can also transform other discrete data based on standard distances (see below). In addition to tree searches under the minimum evolution criterion, algorithmic tree building techniques, like UPGMA and Neighbor Joining (for more details see ”Tree Searching and Reconstruction”) are commonly used to infer phylogenetic relationships with distance data.

11.2 Pairwise Distance Methods

A key difference between parsimony and likelihood methods is that parsimony seeks solutions that minimize the amount of evolutionary change to explain the observed data, whereas maximum likelihood attempts to estimate the actual amount of change according to some explicit model of evolution. This difference is important because as mutations are fixed in a genome, there is an ever-increasing probability of **superimposed changes** occurring at a single sequence position in a set of aligned sequences (also called **multiple hits**). That is, changes at a particular site along a lineage of a phylogeny may mask earlier changes at that site, and parallel or convergent changes may occur at the same site in different lineages. Because of this, parsimony will tend to underestimate the amount of evolutionary change, unless the actual rate of change is extremely low. Failure to account for changes could obviously obscure the evolutionary history of a set of sequences, and thus their relationships.

An alternative to likelihood for minimizing the impact of multiple substitutions is the use of corrected distances that account for multiple hits by estimating the number of unseen changes using the same kinds of models used in maximum likelihood analyses. These corrected distances are then estimates of the true **evolutionary distance**, which reflects the mean number of changes per site that have occurred between pairs sequences since divergence from their most recent common ancestor.

For some data types (e.g., immunological, DNA hybridization), there is no alternative to the use of distance methods. For other types of data, including sequence data (both DNA and protein), restriction site, and allozyme data, distance methods can provide a means by which to take advantage of models of molecular evolution when likelihood methods are either unavailable or intractable. Even with the increase speed of computers and the availability of programs that perform fast maximum likelihood searches, distance methods still remain useful, especially for the analysis of large data sets, where the speed of distance methods allows for the testing of alternative tree topologies.

One negative side to distance methods is that information is lost when transform character data to pairwise distances. For example, Penny (Penny, 1982) has shown cases were several data sets yielded the same distance matrix, but given only the distances, it is impossible to recover the original sequences. Another drawback to to distance analysis is that it does not lend itself to the combination of data (Miyamoto, 1985). Finally, distance based methods do not allow researchers to identify particularly informative characters (or regions) in order to limit future studies to those characters that are most informative.

11.2.1 Standard distances

Two distances are computed for each pair of taxa i and j:

1. The "absolute distance"

$$(11.1) \qquad d(i,j) \quad = \quad \sum_{k \in S} w_k * diff(x_{ik}, x_{jk}),$$

where S is the set of characters that are not "ignored" or excluded, wk is the weight currently assigned to character k, x_{ik} and x_{jk} are the states of character k in taxa i and j, respectively, and $diff(x_{ik}, x_{jk})$ is the cost of a change from state x_{ik} to state x_{jk}. The costs specified by $diff(y,z)$ are determined as follows:

If either y or z is "missing," or if both y and z are "missing," $diff(y,z) = 0$.

Otherwise, for unordered characters,

$diff(y,z) = 1 if y \neq z$, 0 otherwise;

for ordered, Dollo, and irreversible characters,

$diff(y,z) = |y - z|$;

for user-defined character-state trees,

$diff(y,z)$ = the number of branches lying on the path of the character-state tree connecting states y and z;

for stepmatrix characters,

$diff(y,z) = s_{yz}$, the corresponding element of the applicable stepmatrix.

2. The "mean distance"

$$(11.2) \qquad d_m(i,j) \quad = \quad , \frac{d(i,j)}{\sum_{k \in S} w_k^*}$$

where $w_k^* = w_k$ if both x_{ik} and x_{jk} are nonmissing and zero otherwise. That is, the absolute distance is divided by the total weight of the characters for which neither i nor$j$ has the state "missing." Mean distances are more meaningful when some taxa have much higher proportions of missing data than others.

Distance matrix output has the following format:

```
Pairwise distances between taxa


   Below diagonal: Absolute distances

   Above diagonal: Mean distances

                1        2        3        4        5        6        7        8
  1 taxonA      -    0.429    0.357    0.429    0.500    0.857    0.571    0.571
  2 taxonB      6        -    0.357    0.571    0.500    0.429    0.857    0.857
  3 taxonC      5        5        -    0.214    0.286    0.500    0.500    0.500
  4 taxonD      6        8        3        -    0.214    0.571    0.286    0.286
  5 taxonE      7        7        4        3        -    0.500    0.500    0.500
  6 taxonF     12        6        7        8        7        -    0.571    0.571
  7 taxonG      8       12        7        4        7        8        -    0.000
  8 taxonH      8       12        7        4        7        8        0        -
```

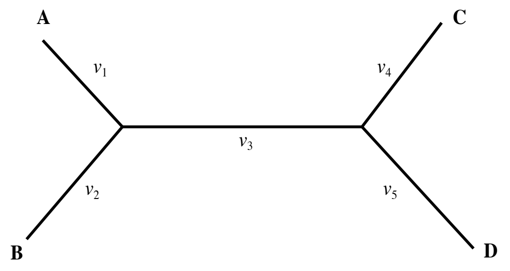
11.2.2 *Additive distances*

If one could determine exactly the true evolutionary distance implied by a given amount of observed sequence difference between each pair of taxa under study, these distances would have a very useful property of **tree additivity** (Figure 10.1): the evolutionary distance between each pair of taxa would be equal to the lengths of each branch lying on the path between the members of each pair. Additive distances satisfy the **four-point metric condition** (Buneman, 1971): for any four taxa A, B, C, and D,

$$d_{AB} + d_{CD} \quad \leq \quad max(d_{AC} + d_{BD}, d_{AD} + d_{BC}) \tag{11.3}$$

where d_{ij} is the distance between i and j, and "max" is the maximum value function. Conceptually, this simply means that of the three sums of distances $d_{ij} + d_{kl}$ where $i \neq j \neq k \neq l$, one of these must be as small or smaller than the other two, and these other two must be equal. For example, in Figure 10.1A:

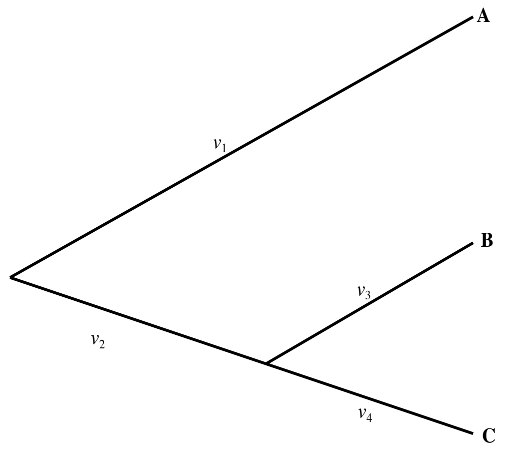
(A)



Additive properties:

$d_{AB} = v_1 + v_2$
$d_{AC} = v_1 + v_3 + v_4$
$d_{AD} = v_1 + v_3 + v_5$
$d_{BC} = v_2 + v_3 + v_4$
$d_{BD} = v_2 + v_3 + v_5$
$d_{CD} = v_4 + v_5$

(B)



Additive properties:

$d_{AB} = v_1 + v_2 + v_3$
$d_{AC} = v_1 + v_2 + v_4$
$d_{BC} = v_3 + v_4$

Ultrametric properties:

$v_3 = v_4$
$v_1 = v_2 + v_3 = v_2 + v_4$

Figure 11.1 *Additive and ultrametric trees. (A) An additive tree relating four taxa: A, B, C, and D. It also lists the relationship between the six taxon-to-taxon distances (d_{AB} through d_{CD}) and five branch lengths (v_1 through v_5). All sets of pairwise distances that satisfy the four-point condition (see text) can be represented as a unique additive tree. (B) An ultrametric tree relating three taxa: A, B, and C. In addition to having additive properties, every common ancestor is equidistant from all its descendants. Thus, the most recent common ancestor of B and C is v_3 from B and v_4 from C, therefore $v_3 = v_4$.*

$$\begin{aligned}
d_{AB} + d_{CD} &= v_1 + v_2 + v_4 + v_5 \\
d_{AC} + d_{BD} &= (v_1 + v_3 + v_4) + (v_2 + v_3 + v_5) = \\
&\quad v_1 + v_2 + v_4 + v_5 + 2v_3 \\
d_{AD} + d_{BC} &= (v_1 + v_3 + v_5) + (v_2 + v_3 + v_4) = \\
&\quad v_1 + v_2 + v_4 + v_5 + 2v_3
\end{aligned}$$

Tree-additive distances can be fitted to an unrooted tree such that pairwise distances are equal to the sum of the lengths of the branches along the path connecting the corresponding taxa (Figure 10.1A). Unfortunately, due to the finite amount of available data, stochastic error will case deviation of the estimated evolutionary distances from perfect tree additivity even when evolution proceeds exactly according to the model of molecular evolution used for the distance corrections.

11.2.3 *Additive-tree methods*

11.2.3.1 Fitch-Margoliash and related methods

11.2.3.2 The minimum evolution method

11.2.4 *Ultrametric Distances*

11.3 *Distance transformation for sequence data*

11.3.1 *Measurement of sequence dissimilarity*

11.3.2 *Accounting for superimposed changes*

11.4 *Setting distance models*

Use the Dset command to set options for distance analysis.

Analysis > Distance Settings...

Example 1:

The following example of the Dset command demonstrates one of the ways to specify a maximum likelihood model when the Distance option is set to ML. On line two the Lscores command is used to estimate all six substitutions rate classes and the gamma shape parameter (α) on the neighbor joining generated from line one. The Lset command is then used to fix the parameter values to those estimated by the Lscores command.

Command-line

- Type: `nj;`
- Type: `lscore 1/nst=6 rmatrix=estimate rates=gamma shape=estimate;`
- Type: `lset rmatrix=previous shape=previous;`
- Type: `dset distance=ml;`

Menu equivalent

- Select **Analysis >Neighbor Joining/UPGMA...**
- Make sure the **Neighbor joining** option is selected under **Algorithm** and click **OK**
- Select **Trees >Tree Scores >Likelihood...**
- Click on **Likelihood Settings...**. Under **Substitution Model**, select **General time-reversible ("6")** and the **Estimate** button. Under the **Among-site Rate Variation** submenu, choose the **Gamma distribution** option in the **Variable sites** box. Select that the **Shape parameter** is **Estimate** and Click **OK**. Click **OK** again
- Next, select **Analysis >Likelihood Settings...**
- Under both the **Among-site Rate Variation** and **Substitution Model** menus, select that the parameters be **Set to: previous** . This will automatically fill in the values for the parameter just estimated in the previous step.
- Select **Analysis >Distance Settings...**
- Under the **DNA/RNA distance:** submenu, select **Maximum Likelihood...**. This will take you back to the "Likelihood Settings" window. Click **OK**, then click **OK** again.

Example 2:

The following example pools transitions into a common class and defines two seperate classes for transversions.

Command-line

- Type: `dset distance=custom class=(a b a a c a);`

Menu equivalent

- Select **Analysis >Distance Settings..**
- Under the **DNA/RNA distance:** submenu, select **Custom(moment estimator)...**.
- You will be presented a interface where you can set the substitution classes. Set **A** ↔ **C**, **A** ↔ **T**, **C** ↔ **G**, and **G** ↔ **T** to **a**; **A** ↔ **G** to **b**; and **C** ↔ **T** to **c**.
- Click **OK**

Example 3:

The following use of the Dset command corresponds to the method of Fitch and Margoliash (1967).

Command-line

- Type: `dset distance=jc objective=lsfit power=2;`

Menu equivalent

- Select **Analysis >Distance Settings..**
- Under the **DNA/RNA distance:** submenu, select **Jukes-Cantor**.
- Under **Other options:**, select **Objective function**
- Check the dialogs **Weighted least squares** and **Inverse-squared weighted (power=2)**
- Click **OK**

Available Options:

- **Distance = User| Total| Mean| Abs| P| JC| F81 | TajNei| K2P| F84| HKY85| K3P| TamNei| GTR | Custom| ML| LogDet| Upholt| NeiLi**
- **Rates = Equal| Gamma**
- **Shape =** ***real-value***
- **PInvar =** ***real-value***
- **RemoveFreq = Proportional| Equal**
- **EstFreq = All| Constant**
- **AllSitesMean = Yes| No**
- **Subst = All| TV| TI| TRatio**

- **Class = (cAC cAG cAT cCG cCT cGT)**
- **BaseFreq = Equal| Empirical**
- **MissDist = Infer| Ignore**
- **Objective = ME| LSFit**
- **Power =** *integer-value*
- **NegBrLen = Prohibit| Allow| SetZero| SetAbsVal**
- **DCollapse = Yes| No**
- **Wts = Repeatcnt| Ignore**

Description of options

Distance =
User|Total|Mean|Abs|P|JC|F81|TajNei|K2P|F84|HKY85|K3P|TamNei|GTR|Custom|ML|LogDet|Uph

Use the Distance option to specify the distance measure used to calculate the scores of the trees in memory. The following distances may be used:

11.4.1 *Distance definitions*

User	user-defined distance (see ”Commands used in the DISTANCES Block”)
Total	number of pairwise character differences
Mean	mean number of pairwise character differences (adjusted for missing data)
Abs	absolute distances
P	uncorrected distance often referred to as the p-distance or dissimilarity (D) distance.
JC	Jukes and Cantor (1969); equal base frequencies, one substitution types
F81	Felsenstein (1981); unequal base frequencies, one substitution type
TajNei	Tajima and Nei (1984); unequal base frequencies, one substitution type
K2P	Kimura (1980); equal base frequencies, unequal Ti:Tv
F84	Felsenstein (1984); unequal base frequencies, unequal Ti:Tv
HKY85	Hasegawa, Kishino, and Yano (1985); unequal base frequencies, unequal Ti:Tv
K3P	Kimura (1981); equal base frequency; 3 substitution types
TamNei	Tamura and Nei (1993); unequal base frequency; 3 substitution types
GTR	general time-reversible model (Lanave et al. 1984, and Rodriquez et al. 1990) ; unequal base frequencies, 6 substitution types.
Custom	user defines the base frequencies and number of substitution types
ML	maximum likelihood estimator of distance under current maximum likelihood settings
LogDet	log-determinant (Steel, 1994; Lockhart et al., 1994) or paralinear (Lake, 1994) distances
Upholt	restriction-site distance of Upholt (1977)
NeiLi	restriction-site distance of Nei and Li (1979)

The following set of options apply only to the calculation of pairwise distances:

Rates = Equal|Gamma

Unless Rates = Gamma, rates across all characters are assumed to be equal. If Rates = Gamma, rates across sites are assumed to follow a continuous gamma distribution. The shape of the gamma distribution is equal to the value specified for the SHAPE option. This option is not available for Abs, P, LogDet, or ML distances. To specify gamma distributed rates for ML distances use the ”LSet” command.

Shape = *real-value*

Use Shape to specify the shape of the gamma distribution used to accommodate among-site rate variation. Again, this option is not available for Abs, P, LogDet, or ML distances. To specify the gamma-shape parameter for ML distances use the ”LSet” command.

PInvar = *real-value*

Use PInvar to specify the proportion of invariable sites (i.e., sites that are unable to accept substitutions); the remaining sites are assumed to vary at the same rate.

The following options are available only if a *real-value* has been specified under PInvar:

RemoveFreq = Proportional|Equal

If RemoveFreq = Equal, the same number of identical sites is removed from each nucleotide in a given pairwise comparison. If RemoveFreq = Proportional, then the number of identical sites removed is proportional to the frequency of each base.

EstFreq = All|Constant

If RemoveFreq = Proportional, then the frequency of each base may be estimated from the entire data set (EstFreq = All) or only the invariable sites (EstFreq = Constant).

AllSitesMean = Yes|No

Adjust the distance so that it represents the mean number of substitutions over all sites, rather than over only the variable sites.

Subst = All|TV|TI|TRatio

By default PAUP* counts or estimates substitutions of all types (Subst = All). For many DNA distances, the distance represents the mean number of substitutions per site that have accumulated since a pair of taxa last shared a common ancestor. You can restrict PAUP* to count or estimate changes representing transition, changes from a purine (A or G) to pyrimidine (C or T) (Subst = TI) or transversion, changes from a purine to a purine and pyrimidine to a pyrimidine (Subst = TV). Finally, PAUP* will output the transition/traversion ratio for the distance chosen when Subst = TRatio.

Class = (cAC cAG cAT cCG cCT cGT)

If Distance = Custom, then substitution types may be assigned to any of six classes. For example, substitution types may be pooled into common classes to reflect changes between structurally similar molecules (e.g. transitions and transversions). A user is free to decide among the many possible ways that substitution types may be divided or combined.

BaseFreq = Equal|Empirical

If Distance = Custom, base frequencies used to estimated pairwise distances may be based on either the empirical base frequency or equal base frequencies (i.e., frequency of each base equals 1/4).

MissDist = Infer|Ignore

Unless MissDist = Ignore , PAUP* infers ”missing-data” sites (including gaps) and ambiguity codes by distributing them proportionately to unambiguous changes. You may also choose to ignore sites for the affected pairwise comparison (Ignore).

The following set of options apply to the calculation of branch lengths and tree scores:

Objective = ME|LSFit

Choose between minimum evolution (ME) and least-squares (LSFit) objective functions. See Swofford et al. (1996) for a discussion of these objective functions.

Power = *integer-value*

Sums of squared deviations in least-squares analyses (Objective = LSFit) may be unweighted (Power = 0), weighted by the inverse (Power = 1) of the distance, or weighted by the squared inverse Power = 2). Power = 2 corresponds with the method of Fitch and Margoliash (1967).

NegBrLen = Prohibit|Allow|SetZero|SetAbsVal

Use NegBrLen to specify negative branch length handling. If NegBrLen = Prohibit, then PAUP* constrains branch lengths to be nonnegative. By default, PAUP* allows but sets to zero negative branch lengths when computing the value of the objective function (SetZero). You may also allow negative branch lengths (Allow) or an absolute value (SetAbsVal) when computing the value of the objective function.

Dcollapse = Yes|No

Use Dcollapse to collapse branches of effectively zero length when searching. If a branch has a length less than or equal to $1x10^{-8}$, it is eliminated from the tree.

Wts = Repeatcnt|Ignore

If Wts = commandIgnore, character weights are ignored. The default (Wts = Repeatcnt) treats integer weights as repeat counts.

11.5 Displaying the distance matrix

Use the ShowDist command to output a matrix of ”distances” between taxa. There are no options for this command. Use the Dset command to specify the distance measure to be calculated.

Syntax: : `ShowDist;`

Menu equivalent: **Data >Show Pairwise Distances**

11.6 Calculating distance score on tree in memory

Use the DScores command to calculate scores of trees in memory according to distance criterion. The tree-list indicates which trees are to be evaluated.

Syntax: : DScores [*tree-list*][/*options*];

Menu equivalent: **Trees >Tree Scores >Distance ...**

Available Options:

- *** SortTrees = No| Yes**
- **ScoreFile =** ***file-name-for-scores***
- *** Replace = No| Yes**
- *** Append = No| Yes**
- *** DisplayOut No| Yes**
- **Distance = User| Total| Mean| Abs| P| JC| F81 | TajNei| K2P| F84| HKY85| K3P| TamNei| GTR | Custom| ML| LogDet| Upholt| NeiLi**
- **Rates = Equal| Gamma**
- **Shape =** ***real-value***
- **PInvar =** ***real-value***
- **RemoveFreq = Proportional| Equal**
- **EstFreq = All| Constant**
- **AllSitesMean = Yes| No**
- **Subst = All| TV| TI| TRatio**
- **Class = (cAC cAG cAT cCG cCT cGT)**
- **BaseFreq = Equal| Empirical**

- **MissDist = Infer| Ignore**
- **Objective = ME| LSFit**
- **Power =** ***integer-value***
- **NegBrLen = Prohibit| Allow| SetZero| SetAbsVal**
- **DCollapse = Yes| No**
- **Wts = Repeatcnt| Ignore**

Description of options

SortTrees = No|Yes

When multiple trees are evaluated under the Dscores command SortTrees = Yes will sort tree scores so that they are displayed in ascending order.

ScoreFile = *file-name-for-scores*

If ScoreFile is specified, distance scores under the current objective function for each tree evaluated is saved to a text file. The *file-name-for-scores* must follow the conventions described under "Input/Output files."

Replace = Yes|No

Append = Yes|No

If the *file-name-for-scores* already exists, you will be prompted for confirmation before the existing file is replaced. Use Replace = Yes to suppress this prompt; the existing file will be quietly overwritten by the new data. Alternatively, you may specify Append, in which case a new Trees block will be concatenated to the end of an existing file.

DisplayOut No|Yes

SET DISPLAYOUT = Yes if you wish to suppress output to the display buffer.

The remaining options for this command are the same as those used under the DSet command.

11.6.1 Saving distances

Use the SaveDist command to save distance matrix to a file.

Data > Save Distances To File...

12 Options And Subcommands Affecting Multiple Commands

Several options apply to a number of commands. Specification of these options on one command affects all commands that use the same options. To minimize redundancy, these options are described in the following sections rather than in the description for each individual command to which the option applies.

12.1 Tree-searching options

These options pertain to the commands that request searching for trees.

Keep = *real-value*

If real-value is zero, only the best trees found will be saved. If real-value $\geq$ zero, all trees of score $\leq$ real-value will be saved. By default, real-value = 0, so that only the best trees found will be saved.

Enforce = Yes|No

Enforce requests that topological constraints be enforced; i.e., trees that are not compatible with the constraint tree are not evaluated. If the Constraints option (see below) is not used to specify a constraint tree, the ”current” constraint tree is used.

Dstatus = *integer-value*|None

Information on the progress of the search (number of trees examined, number of trees saved, etc.) may output to the display buffer using the Dstatus setting. Specify an *integer-value* to control how frequently (in seconds) information is output to the display or specify Dstatus = No to suppress this information. The default setting for the Macintosh versions of PAUP* is Dstatus = None and Dstatus = 60 for all other versions.

Constraints = *constraint-tree-name*

The specified constraint tree, which must have been defined in a previous Constraints command, becomes the current constraint tree. You must also specify Enforce if you want to search under constraints.

Converse = Yes|No

Converse is specified in conjunction with Enforce, only trees that are not compatible with the constraint tree are evaluated. Converse = No reverses the effect of a previous Converse specification.

Status = Yes|No

Information on the progress of the search (number of trees examined, number of trees saved, etc.) is output to a status window. Status = No suppresses this window.

12.2 Tree-rooting options

Root = OutGroup|Lundberg|Midpoint

The Root option is used to specify how unrooted trees are to be rooted prior to output. You can choose OutGroup rooting, using whichever outgroup you have selected; Midpoint rooting, which roots the tree at its midpoint; or Lundberg rooting, which requires that a previous AncStates command has been issued. By default, OutGroup rooting is in effect.

OutRoot = Polytomy|Paraphyl|MonoPhyl

If outgroup-rooting is currently selected, there are three options for output. The outgroup can make up a polytomy next to the ingroup (Polytomy, the default); or it can be made to be paraphyletic relative to the ingroup (Paraphyl); or the monophyletic sister group to the ingroup (MonoPhyl).

12.3 Tree output options

TCompress = Yes|No

Specify TCompress to output tree diagrams in a ”vertically compressed” format. The resulting diagram is not as aesthetically appealing, but it allows more of a large tree to be seen on one screen (or in one window), and it takes less paper to print.

12.4 Options for character-matrix listings

ShowExcluded = Yes|No

Unless ShowExcluded is specified, ”excluded” characters are not shown in character-matrix listings.

CMLabels = Yes|No

By default, character names are used to label the columns of character-matrix listings. If you want to use numbers even when character names are available, specify CMLabels = No.

CMCStatus = Yes|No

If CMCStatus is specified, characters that are constant, ”zapped,” uninformative, or excluded are identified by asterisks at the top of each column of a character-matrix listing.

CMColWid = *column-width*

The value specified for *column-width* determines the number of columns used for each character in the data matrix. The default is CMColWid = 2, so that one blank column appears between each column of character state data. For sequence data, you may want to use CMColWid = 1 in order to fit more characters onto each line of output.

CMShowEq = Yes|No

Unless CMShowEq has been specified, if the possible state assignments to an interior node correspond to a multistate taxon code specified in an Equate macro, the corresponding Equate character is shown rather than the equivalent set of character states.

12.5 Other options

Opt = AccTran|DelTran|MinF

The Opt = *subcommand* determines how the characters are optimized on the tree(s) in memory. AccTran (the default) uses ”accelerated transformation”, DelTran uses ”delayed transformation”, while MinF optimizes so as to minimize the ”f-value” (Farris, 1972; Swofford and Maddison, 1987). See the section on character-state optimization for detailed discussion of these options.

AncStates = *ancestral-states-name*

Change the ancestor currently in effect to the ancestral-states-name defined in an earlier Ancstates command (or to Standard). This option affects searching and character-state reconstruction algorithms.

12.6 The SET command

Many of the above commands are controlled with the Set command. The Set command is used to set a variety of options whose scope extends beyond any single command.

Syntax: : `Set` [*options*];

Menu equivalent: **Options**

Available options

- Root = Outgroup|Lundberg|Midpoint
- InitSeeds = 0|1
- Monitor = Yes|No
- SemiGraph = Yes|No
- MaxTrees = *integer-value*
- Increase = No|Prompt|Auto
- AutoInc = *integer-value*
- ShowExcluded = Yes|No
- AllowPunct = Yes|No
- CMLabels = Yes|No
- CMShowEq = Yes|No
- CMColWid = *integer-value*
- CMCstatus = Yes|No
- Constraints = *constraint-name*
- ErrorBeep = Yes|No
- QueryBeep = Yes|No
- KeyBeep = Yes|No
- NotifyBeep = Yes|No
- ErrorStop = Yes|No

- AutoClose = Yes|No
- TOrder = Standard|Right|Left|Alphabet
- CheckEvts = Yes|No
- TCompress = Yes|No
- ShowTaxNum = Yes|No
- Background = Yes|No
- Status = Yes|No
- OutRoot = Polytomy|Paraphyl|Monophyl
- VisNotif = None|ShowAlert|FlashOnly
- FlushLog = Yes|No
- AllowEnd = Yes|No
- AllDigLab = Prohibit|Warn|NoWarn
- Criterion = Parsimony|Likelihood|Distance
- DStatus = *integer-value*|None
- DefaultMode = Yes|No
- StoreBrLens = Yes|No
- StoreTreeWts = Yes|No
- WarnReset = Yes|No
- WarnTree = Yes|No
- WarnTSave = Yes|No
- WarnBlkName = Yes|No
- WarnRoot =Yes|No
- WarnRedef = Yes|No
- ShowAbbrev = Yes|No
- Pause = No|Silent|Beep|Msg
- TaxLabels = Full|Truncate
- DropMode = Edit|Conditional|Execute

Description of options

Monitor = Yes|No

Monitor = No suppresses output to the ”main display” (window or terminal screen) and is useful when you want to send output to the log file and/or printer only. Monitor = Yes reactivates the main display.

At least one output destination must be active at all times. Consequently, if no log file is active or the ”echo to printer” (Echo) option is not set, output will be sent to the main display even if Monitor = N0 has been requested.

InitSeeds = 0|1

A few commands used in PAUP* require sequences of ”random” numbers (i.e., Bootstrap, Jackknife, Hompart, Permute, RandTree, Puzzle, and SurfCheck). Other command also use ”random” numbers, but only when certain options are specified (i.e., GenerateTrees, HSearch, NJ, and StarDecomp). PAUP* uses a linear congruential method starting with an integer between 1 and 2,147,483,646 to seed a random number generator. By default InitSeeds = 0, instructs PAUP* to obtain the initial seed from the system clock. User may also set InitSeeds=1, in which case all seeds are initialized to 1. If you do not specify a starting seed for subsequent runs, the seed defaults to the next number in the random number sequence initiated during the previous run. Users are always given the option to specify a value for the seed on any command that accepts seed settings, thus overriding the PAUP* default of referencing the system clock. This InitSeeds option ONLY applies if a seed value has not been initialized during the current run of the program. You can reset the sequence using a clock-based seed value by specifying a seed value of 0 on any command that accepts seed settings.

AllowPunct = Yes|No

Unless ALLOWPUNCT is specified, taxon and character names in input data matrices must conform to the NEXUS specification; namely they cannot contain special punctuation characters like parentheses, hyphens, etc. (see ”Identifiers”). If ALLOWPUNCT = YES is set prior to processing of a DATA or CHARACTERS block, then taxon and character names are allowed to contain these special characters. This option applies only to the DATA and CHARACTERS blocks; names containing special characters must be enclosed within single quotes in other contexts. This option allows backward compatibility with files prepared for PAUP 3.1; new data files should always conform to the NEXUS standard to prevent incompatibilities with other programs.

SemiGraph = Yes|No

PAUP* uses special characters in its internal font to draw trees and other items. On the IBM-PC, these characters are nonstandard ″high ASCII″ characters. On the Macintosh, these characters are neither in the standard 128 ASCII characters nor in the set of special characters normally included with Macintosh fonts. Thus, although the trees look nice when drawn in the main display window, they may not look right when printed on some printers. Therefore, PAUP* ordinarily translates these ″semigraphics″ characters to standard ASCII substitutes when output is directed to a printer, file, or document window. If you want to override this behavior, specify SemiGraph. (E.g., many IBM-PC printers can print the high ASCII characters, and the Apple LaserWriter can create a ″bit map″ version of PAUP*'s internal ″PAUPMonaco″ font).

This option is relevant only for DOS, Windows, and Macintosh interfaces.

AutoClose = Yes|No

After a search has ended the status window will remain in the foreground until the user closes it. Use AUTOCLOSE = YES to close the status window automatically after a run has ended.

NotifyBeep = Yes|No

Unless NOTIFYBEEP = NO, a ″beep″ is sounded at the end of a search.

TOrder = Standard|Right|Left|Alphabet

Specifies the convention used to ″order″ the tree.

FlushLog = Yes|No

Specification of FLUSHLOG causes the file's buffer to be flushed after every line of output. Ordinarily, this degrades system performance and is not recommended. However, there may be situations in which immediate flushing is useful.

DropMode = Edit|Conditional|Execute

The DROPMODE option allows the user to always edit (EDIT) or always execute (EXECUTE) when a PAUP* document is dragged onto the application icon or PAUP*'s main window. Otherwise, under the default setting (DROPMODE =CONDITIONAL) the file will be executed if no active data file has been set. If a data file is already active, the file will be edited. The DROPMODE option is only available to the Macintosh and Windows interfaces.

AllowEnd = Yes|No

The NEXUS convention is to terminate blocks with END; however, earlier versions of PAUP used ENDBLOCK. Unless ALLOWEND = NO, PAUP* will accommodate both character strings for block termination.

AllDigLab = Prohibit|Warn|NoWarn

By default, PAUP* issues a warning if all digit taxa or character labels are included. Use ALLDIGLAB to prohibit the use of all-digit labels (PROHIBIT) or to allow all-digit labels and suppress warnings (NOWARN). See "Identifiers" for a description of valid taxa and character labels.

Criterion = Parsimony|Likelihood|Distance

Use CRITERION to selection from the three optimality criterian.

DefaultMode = Yes|No

By default when option settings or other information is required, a dialog box will be opened. If DEFAULTMODE = YES, then PAUP* will take a default action. Because the default action will not be appropriate in every situation, this option is not recommended unless you know what affect the default action will have on your analyses.

StoreBrLens = Yes|No

If branch lengths are present in the TREES block, then STOREBRLENS = YES stores branch lengths for subsequent use within the program. Otherwise, the branch lengths are discarded.

StoreTreeWts = Yes|No

If tree weights are present in the TREES block, then STORETREEWTS = YES stores tree weights. When tree weights are the reciprocal of the number of trees found in either a BOOTSTRAP or JACKKNIFE replicate, this option allows the combination of bootstrap results from runs performed at different times or on different machines and the recovery of results obtained prior to a system crash. (See ”UseTreeWts” for more details)

ShowAbbrev = Yes|No

Specifies whether help shows command and option names so as to indicate the shortest acceptable abbreviation.

TaxLabels = Full|Truncate

If TAXLABELS = FULL, then the full 127 characters of long taxon labels will be output to the display buffer (although they will still be truncated somewhat if the total output width cannot accommodate the long names). Otherwise, taxon names are truncated to 16 characters.

ErrorStop = Yes |No

Ordinarily, PAUP* stops processing an input file when unrecognized commands, Options-command keywords, or formats are encountered. If ERRORSTOP = NO is specified, a warning message is issued and processing is allowed to continue.

WarnReset = Yes|No

Ordinarily, PAUP* issues a warning message when an input file containing a DATA block is executed and a DATA block has already been processed. Specification of WARNRESET = NO suppresses this warning.

WarnTree = Yes|No

Use WARNTREE to issue a warning about treefile operations that could delete unsaved trees.

WarnTSave = Yes|No

Use WARNTSAVE to issue a warning before quitting if there are unsaved trees in memory.

WarnBlkName = Yes|No

Use WARNBLKNAME to issue a warning when unrecognized blocknames are included in a data set.

WarnRoot = Yes|No

Use WARNROOT to issue a warning about the rooting status of user-input trees.

WarnRedef = Yes|No

Use WARNREDEF to issue a warning about redefining names for sets, constraints, etc.

Pause = No|Silent|Beep|Msg

By default, output is displayed to the display buffer without pause. Alternatively, PAUP* will display one screen of output at a time either pausing silently (SILENT), issuing a warning beep (BEEP), or printing a message to the screen (MSG) for the user press return key before displaying the next screen.

ShowTaxNum = Yes|No

When SHOWTAXNUM = YES trees are displayed with both taxa labels and taxa numbers. The number are based on the order in which the taxa appear in the CHARACTERS or DATA blocks.

The following three options affect the setting of the maximum number of trees that PAUP* can store at any given time:

MaxTrees = *integer-value*

The MAXTREES parameter specifies the maximum number of trees that can be saved. Setting MAXTREES to a large value will reduce the likelihood that the tree buffer will become full during a search or tree-file operation, at the expense of a larger chunk of memory being tied up and therefore unavailable for other purposes.

Ordinarily, if the number of trees found during a search reaches the value of MAXTREES, you will be given a chance to increase MAXTREES before proceeding. This behavior can be altered using the Increase option (see below). MAXTREES is initially set to 100.

The MAXTREES option apply to the entire search in random-addition-sequence searches. If you would like to impose a maximum tree limit for each addition-sequence replicate then you must use the NCHUCK and CHUCKSCORES options under the HSEARCH command. For replicated searches such as the bootstrap (BOOTSTRAP), the jackknife (JACKKNIFE), and the partition-homogeneity test (HOMPART) the limits are always applied separately for each search.

Increase = Prompt|Auto|No

The setting of the Increase option determines the action taken by PAUP* if the limit on the number of trees that can be stored (=MAXTREES, see above) is reached during a search or a tree-file operation. If INCREASE = PROMPT, you will be given the opportunity to increase MAXTREES. If INCREASE = AUTO, MAXTREES will automatically be increased by a number of trees equal to the current AUTOINC setting (see below). If INCREASE = NO, MAXTREES will not be increased, and no prompt will be issued. In this case, a ”tree-buffer overflow” occurs which can affect the effectiveness of the search in progress. (The tree-buffer overflow condition will be documented in the output.)

AutoInc = *integer-value*

The AUTOINC value species the number of trees by which MAXTREES is increased when the number of trees saved reaches MAXTREES and the INCREASE = AUTO option is in effect. AUTOINC is initially set to 100.

Example 1

In the following example, a maximum tree limit of 100 for each random-addition-sequence replicate is set. The NCHUCK and CHUCKSCORE options of the HSEARCH command are used to set the limit for each replicate search.

Command-line

- Type: `set maxtrees=1000 increase=no;`
- Type: `hsearch addseq=random nreps=10 nchuck=100 chuckscore=1;`

Menu equivalent

- Select **Options >Set MaxTrees...**
- Enter `1000`, check the **Leave unchanged, and don't prompt** dialog and click **OK**.
- Select **Analysis >Heuristic Search...**
- Under the **Stepwise-Addition Options**, Select **random** and enter **# reps** = `10`
- Under the **Branch-Swapping Options**, check **Save no more than**, and enter `100` **trees** ≥ **score** `1` **(each rep)**

The following three options specify whether PAUP* sounds a ”beep” when various kinds of errors occur:

ErrorBeep = Yes|No

Ordinarily, PAUP* beeps to alert you that an error message has been issued. Specify ERRORBEEP = NO to suppress these beeps.

QueryBeep = Yes|No

Ordinarily, PAUP* beeps to alert you when it stops for your input before it can continue a process. Specify QUERYBEEP = NO to suppress these beeps.

KeyBeep = Yes|No

Ordinarily, PAUP* beeps when you type a key that is invalid in the current context. Specify KEYBEEP= NO to suppress these beeps.

The following three options are specific to the Macintosh version.

Background = Yes|No

Ordinarily, PAUP* continues processing when it is moved to the background. Specify BACKGROUND = NO to suppress background processing, thereby giving more time to the foreground application.

CheckEvts = Yes|No

CHECKEVTS = NO disables ”event-checking,” causing all mouse clicks an key presses to be ignored.

Speed of PAUP* searches is improved somewhat, but it will not be possible to stop the search (without restarting the computer) or to switch to a different application under MULTIFINDER.

VisNotif = None|ShowAlert|FlashOnly

When a background job ends an alert box will be sent to the foreground and the PAUP* icon on the task-menu will flash. Use VISNOTIF to suppress the alert box (FLASHONLY) or both of the alert box and the flashing PAUP* icon (NONE).

The following options are described in ”Options And Subcommands Affecting Multiple Commands”:

- Constraints = constraint-tree-name
- Status = Yes|No
- DStatus = integer-value|None
- Root = OutGroup|Lundberg|MidPoint

- OutRoot = Polytomy|Paraphyl|Monophyl
- TCompress = Yes|No
- ShowExcluded = Yes|No
- CMShowEq = Yes|No
- CMLabels = Yes|No
- CMCStatus = Yes|No
- CMColWid = integer-value
- ShowAnc

13 Additional commands

Several commands in PAUP* allow the user to interact directly with the operating system (Unix and OSX). These commands do not manipulate trees or data, but provide convenient ways for the user to take advantage of the command line. These options are not available when using the graphical user interface (GUI) with menu driven commands.

13.1 !

Use the ! command to execute a UNIX command from within PAUP*.

You can type a command that contains a semicolon by enclosing the entire command within single-quotes. You can open a temporary UNIX shell by typing, for example, `!csh` or `!sh`. When you are ready to resume your PAUP* session, type exit or Ctrl-D to exit the shell and return to PAUP*. Note, some shells respond differently to this command.

Syntax: : ! [unix-command] ;

13.2 CD

Use the CD command to change the current working directory for the Windows, Unix and DOS versions.

Syntax: `: CD <new-directory>;`

Entering the command with no arguments simply reports the current directory.

14 Parsimony

14.1 Tree Lengths and Character Weights

Parsimony analysis operates under the assumption that characters are independent of each other, so that the length L of a full tree can be computed as the (weighted) sums of the lengths l_j of the individual characters:

$$L = \sum_{j=1}^{C} w_j l_j, \tag{14.1}$$

where C is the total number of characters. Consequently, for the purpose of computing tree lengths, we can treat each character in isolation from the rest. It is this property that allows the "mixing and matching" of character types. For example, nothing prevents you from assuming that one subset of the characters is ordered (Wagner), another is unordered, and yet another Dollo. For any tree, the minimum length required by each character is evaluated under the assumptions assigned to that character and the full length of the tree is obtained by summing over characters.

Typically, $w_j = 1$ for all characters ("equal weights"). However, you may choose any vector $\mathbf{w}$ of weights. For instance, if you had reason to believe that two characters were tightly coupled for reasons other than phylogenetic history, then weighting each of the characters 1/2 would be appropriate. Alternatively, you may have prior knowledge (or at least be willing to assume) that some characters are more reliable than others. These characters could be given higher weight than the rest.

Sometimes the number of states recognized for a character is highly arbitrary. This would be the case, for example, if a character varying continuously in size were broken into a set of discrete states. The character could perhaps be broken into four states (a ″course-grained″ approach) or ten states (a more ″fine-grained″ coding). Remember that under ″equal weighting″ the character will actually have three times as much influence on the analysis under the ten-state coding than it would under the four-state coding [(10-1)/(4-1)]. Likewise, under the ten-state coding, the character would have nine times as much influence on the analysis as an ″equally weighted″ binary presence-absence character. PAUP* provides an option for ″scaling″ character weights so that the total influence of each character is the same, regardless of the number of states. Specifically, a binary character is weighted 1, a three-state character 1/2, a four-state character 1/3, and so on.

NOTE: Some workers have suggested that the problem of the above paragraph can be circumvented by breaking the multistate character into a set of additive binary characters, but this is not the case. If you break a ten-state character into nine binary characters, it is indeed true that each of the resulting binary characters has the same weight as does a truly binary (e.g., presence-absence) character, but there are now nine of them. In fact, for a linearly ordered multistate character, it makes no difference to the analysis whether you treat it as a single character or break it into a set of additive binary characters. Doing the latter serves no useful purpose and merely complicates the interpretation of the output.

Note that most implementations of PAUP* do not allow fractional or decimal character weights. Thus, to assign a weight of 1/2 to a character, you must instead assign a weight of two to all of the other characters. (The reason for this design decision is that integer arithmetic is vastly faster on microcomputers than is floating-point or ″real″ arithmetic.) When scaling weights, it is desirable to choose a smallest common multiple of the desired weights in order to avoid roundoff error. For example, if your data set contains a mixture of two-, three-, four-, and five-state characters and you want to scale weights, use 60 as the ″base″

weight so that the weights 1/2, 1/3, 1/4, and 1/5 correspond to 30, 20, 15, and 12, respectively.

PAUP* also implements the successive approximations character method (Farris, 1969) as implemented in the Hennig86 program (Farris, 1988). This method is discussed in more detail in the section "*A Posteriori* Character Weighting."

14.2 General Concepts

PAUP is a program for inferring phylogenies from discrete-character data under the principle of maximum parsimony. Parsimony methods search for minimum-length trees [*]: trees that minimize the amount of evolutionary change needed to explain the available data under a prespecified set of constraints upon permissible character changes. The best known discrete-character parsimony method, often called Wagner parsimony (Kluge and Farris, 1969; Farris, 1970) treats binary or ordered multistate characters and permits free irreversibility. Multistate characters may also be left unordered (i.e., any character state is permitted to transform directly into any other state), sometimes called Fitch parsimony after Fitch (1971). Other parsimony variants place additional restrictions on the types of character-state changes that are allowed. Dollo parsimony (Farris, 1977) permits each derived, or apomorphic, character state to originate only once. Camin-Sokal parsimony (Camin and Sokal, 1965) prohibits reversals from a derived state to a relatively more ancestral, or plesiomorphic, condition. Also introduced with version 3.0 of PAUP is a procedure based on the work of David Sankoff and his collaborators (Sankoff and Rousseau, 1975; Sankoff and Cedergren, 1983) that allows the user to specify the cost associated with a change from each character state to each other state. Each of these methods is discussed in more detail under "Character Types."

[*]Some authors [e.g., Wiley (1981); Nelson and Platnick (1981)] prefer to distinguish between the terms *tree* - a hierarchical statement regarding genealogical (ancestor-descendant and sister-group) relationships-and *cladogram*- a branching diagram depicting patterns of character distribution or nested sets of synapomorphies. This distinction is purely terminological and probably inappropriate (Hendy and Penny, 1984). PAUP may be used to construct to construct either trees or cladograms, the only difference being how the user interpets the output. "Tree" will be used to refer to both evolutionary trees and cladograms thoughout this manual, with apologies to those who find this usage unacceptable.

Minimization of the total tree length is equivalent to seeking trees that imply the least amount of homoplasy, or similarity not directly attributable to common ancestry. Homoplasies or ”extra steps”-reversals, parallelisms, and convergences-constitute *ad hoc* assumptions required to bring observations into conformity with the ”simpler” explanation that possession of the same character state in two or more taxa is due solely to inheritance from a common ancestor. A word about the relationship between computerized minimum-length-tree approaches and the manual methods used in traditional cladistics [e.g., Hennig (1966); Wiley (1981)] is in order. The latter (”Hennigian”) methods place a heavy emphasis on *a priori* assessment of character polarity, the specification of which of the observed character states represents the ancestral condition in the group under study. *If* character polarities could always be reliably assessed, and *if* there were no homoplasy in the data, then phylogeny reconstruction would amount to nothing more than grouping taxa according to shared derived character states (***synapomorphies***), with no relevance being attached to the sharing of ancestral states (***symplesiomorphies***). That is, all of the taxa that possess a particular derived character state could unambiguously be interpreted as descendants of the ancestor in which the state originated and taxa sharing the ancestral condition could be definitively excluded from that group. Inevitably, however, character conflicts or incompatibilities arise. For example, consider the data shown below:

Table 1. Hypothetical data for example discussed in text.

	Character			
Taxon	1	2	3	4
One	1	1	1	1
Two	1	0	1	1
Three	0	1	0	1
Four	0	0	0	0

For now, we will assume that 0 represents the ancestral state for each character. Thus, we observe that taxa *One* and *Two* share the derived state for characters 1 and 3, while taxa *One* and *Three* share the derived state for character 2. Consequently, if *One* and *Two* were, in reality, ***sister taxa*** (tree A below), the possession of state 1 for character 2 in *One* and *Three* would be a homoplasy, whereas if *One* and *Three* were sister taxa, the derived characters 1 and 3 shared by *One* and *Two* would be homoplasies (tree B below). †

†Of course, a third possibility is that neither *One* and *Two* nor *One* and *Three* represent sister taxa, with all shared derived states being homoplasies.

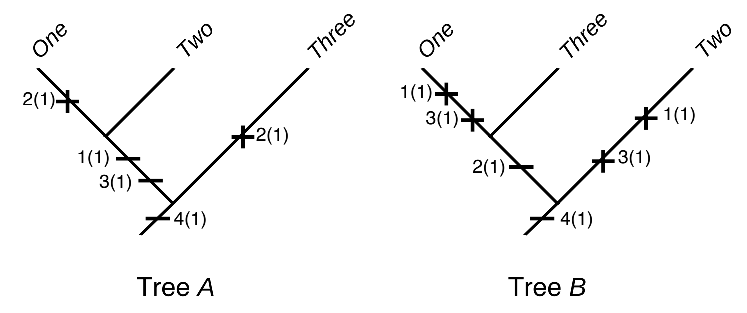


Figure 14.1 *Two trees for the data of Table 1.*

Faced with such a conflict, a practitioner of traditional manual methods would usually decide between these two alternative hypotheses of relationship by invoking the parsimony criterion and therefore choose tree A, which requires fewer assumptions of homoplasy. But this choice is exactly the one that would have been made by the computer method as well, since the tree requiring the least homoplasy would also be the shorter tree (tree A = 5 steps; tree B = 6 steps). Clearly, there is no real difference between computerized methods and manual methods for choosing trees under the parsimony criterion. Two factors can cause the methods to differ in actual practice, however. The first lies in subjectivity introduced by the investigator, who may prefer a tree requiring two or three extra steps in a character assumed to be unreliable or "plastic" over a tree requiring a single extra step in a "better" character. The second is that the human brain is unable to deal effectively with the complexity of large and/or noisy data sets. The above example also illustrates one reason why excessive concern with a priori assessment of character polarities is unwarranted. When incongruent character distributions occur as in this case, the sharing of a derived character state among two or more taxa need not indicate close relationship of those taxa. The problem is accentuated when the outgroup being used for the polarity assessment is heterogeneous. In such cases, parsimony can provide the basis for assigning character polarities, but the most parsimonious state assignment(s) for the most recent common ancestor of the ingroup depends upon the overall structure of the tree (Farris, 1982), including the relationships of the outgroup taxa (Maddison et al., 1984). Thus, the only fully satisfactory recourse is to infer the topology of the tree and the character polarities simultaneously, rather than going through the two-stage process of assigning polarities first and then estimating the tree.

Note: The concepts of character order and character polarity are often confused. The former defines the nature of permitted character-state transformations, whereas the latter refers to the direction of character evolution. Specifying how character states are ordered with respect to each other is not the same as determining which character state is ancestral.

For further amplification of these points, see the sections entitled ”Parsimony Character Types” and ”Outgroups, Ancestors, and Roots” in Chapter 2.

14.3 Searching for Optimal Trees

If the length of a tree under a particular set of assumptions (character types and weights) is less than or equal to the length of any other possible tree for the same data, the tree is said to be optimal (most parsimonious). Depending on the data set, there may be a single most parsimonious tree or two or more (sometimes many) equally parsimonious trees. Ordinarily, the goal of a search is to find all of the equally parsimonious trees that exist for a particular data set under the chosen assumptions. In some cases, you may also be interested in near-optimal trees, i.e., trees whose length is less than or equal to some specified cutoff criterion.

PAUP provides two basic classes of methods for searching for optimal trees, exact methods and heuristics. Exact methods guarantee to find the optimal tree(s) but may require a prohibitive amount of computer time for medium- to large-sized data sets. Heuristic methods do not guarantee optimality but generally require far less computer time.

Note that the search algorithms described below consider only binary trees. In many cases, however, the data will not be sufficient to determine a fully resolved (i.e., binary) tree, however, so that branches of zero-length might occur. In earlier versions of PAUP, two or more trees that could be derived through different resolutions of ***polytomies*** were always considered to represent distinct binary trees containing zero-length branches, with the result that many more trees were sometimes saved than was actually necessary. For example, if a tree contained two trichotomies, nine equally parsimonious trees would be saved-three resolutions for the first trichotomy times three resolutions for the second. Version 3 of PAUP provides an option to collapse branches having zero length to yield polytomies. Because two or more distinct binary trees may collapse to the same nonbinary tree, the program then must ensure that after collapsing, a tree does not become identical to a tree already saved that was obtained by collapsing a different binary tree.

See ”Zero-Length Branches” and Polytomies below for further details on this point.

15 Maximum Likelihood

POWER USER TIP

text here

by Paul O. Lewis, John P. Huelsenbeck, and David L. Swofford

This chapter provides general background on the concepts underlying the use of the method of maximum likelihood for inferring phylogeny in PAUP*. Specific information on how to use the program follows in later chapters. The likelihood models implemented in PAUP* apply only to nucleotide sequence data, and thus no attempt has been made in what follows to explain likelihood models for morphological, continuous, or protein sequence data.

Glossary of terms related to substitution models:

MUTATION: The term mutation is a change in a single copy of a particular DNA sequence. The fate of most mutations is extinction, especially if the change negatively affects the fitness of the individual carrying it. Mutations can spread through a population if they have a selective advantage, or if the effective population size is small enough that the new mutation becomes fixed in a subpopulation due to genetic drift.

SUBSTITUTION: Substitution refers to a mutation that has managed to become fixed in a population. Thus, substitution rates are substantially lower than mutation rates, since the fate of most mutations is extinction rather than fixation.

TRANSITION: The term transition generally refers to a specific type of change, say a base A changing to a T at a given site. This is the usage that predominates in this chapter. Thus, a transition equation gives the probability of a particular change, or transition, occurring in a specified length of time. The term transition can also be used in a more specific sense to refer to a change from one purine to the other purine, or one pyrimidine to a different pyrimidine. When this meaning is intended, we use instead the term ”transition-type substitution” for clarity.

BRANCH LENGTH: Branch length in this chapter refers to the expected number of substitutions per site along a branch of a phylogenetic tree. This is the product of the per site substitution rate and the amount of time the lineage represented by the branch has been evolving. Other definitions of branch length are common, such as the minimum number of substitutions that must be inferred along a branch (the parsimony branch length), or the probability that a difference is observed across the branch. Note that the probability of a difference across the branch, being a prohability, must be between 0 and 1, while the expected number of substitutions can range from 0 to infinity. The term ***edge length*** is synonymous with branch length.

INSTANTANEOUS RATE: This term refers to the rate at which transition probabilities change during an instant of time. As time goes progresses, the probability that no change will be observed across a branch decreases, while the probability of observing a different state increases. The rate of this change is determined by the instantaneous rate, and these instantaneous rates form the heart of any substitution model. It is in the instantaneous rate matrix where biology is incorportated into the substitution model.

15.1 Using likelihoods to compare models

Suppose you were told someone had rolled twenty dice and all had come up one. Most people will register a certain amount of surprise upon observing this data. Suppose further you were more than a little suspicious of this result and asked to see the dice used and found that all of them had a one on each of the six sides! Now how surprised are you that all twenty came up one? You were very surprised at first because the chance of twenty normal dice all coming up one is very small (actually, it is 1 in 3.6 million billion!). You were not surprised at all after observing the dice because you discovered that your underlying die model was wrong, and, with the correct die model, the chance of twenty ones is 1.0. This example, simple as it is, illustrates two important features of probabilities. First, a model is necessary in order to calculate a probability. The probability of twenty dice all coming up one is near zero for the ”normal-die” model, whereas it is exactly 1.0 for the ”ones-on-every-side-die” model. The take-home message here is that the probability of the data can change if the model is changed. Second, the probability of the data may be used to choose which model best fits the data. In this case, clearly the ”ones-on-every-side-die” model provides a more reasonable explanation of the data than the model postulating a normal die. The likelihood of a particular model is defined as the probability of the data given that model. In the example above, the degree to which we were surprised when we considered the data was directly related to the underlying model we assumed. Likelihoods simply provide a quantitative measure of surprise, and become more and more useful (as a means of distinguishing among rival hypotheses) as the data become more and more complex.

The term model was used above in its broad sense as the sum total of everything that must be assumed in order to compute the probability of the data. A narrower definition of the term model is more typical, however. A certain number of assumptions must be made in order to calculate the likelihood. Some of these assumptions we are quite comfortable with and we are satisfied to take these as given. Others, however, we are more concerned about and the testing of these assumptions may become the focus of our investigation. In phylogenetic inference, for example, we may be fairly comfortable with the assumption that the phylogenetic history of the group we are studying can be represented in the form of a dichotomously-branching tree, but much less comfortable in assuming a specific branching pattern (topology) for that tree. In fact, our goal is usually to estimate the topology of the tree, and we therefore treat the various possible topologies as hypotheses to be tested. By convention, those assumptions that we take as given in a particular analysis are collectively referred to as the ***model***, whereas the remaining assumptions needed to calculate the likelihood form the ***statistical hypothesis*** (Edwards, 1972). A model used in maximum likelihood phylogenetic inference normally includes a description of the substitution process (called the ***substitution model***) in addition to the assumption of the binary tree representation mentioned above. In order to completely describe the substitution process, the substitution model must specify the relative rates at which all possible changes from one base to another occur in any single instant of time, dt (the ***instantaneous rates***). From this matrix of instantaneous rates, we can derive a ***transition matrix*** that provides the probabilities of all possible changes from one base to another over a much larger time period, t.

15.2 Model vs. hypothesis

The composition of the model and the hypothesis can change depending on the purpose of the analysis. Sometimes there is reason to call into question the validity of our model. In the case of the tricky dice, above, we were so surprised at the data that we had reason to question whether all aspects of our ”normal die” model was correct. For an example in phylogenetics, there has been much recent interest in testing the adequacy of substitution models (Goldman, 1990; Yang et al., 1994). Since the purpose of such an analysis is to find whether or not a particular substitution model adequately fits the data, we can no longer consider this description of the substitution process to be given and it becomes part of the statistical hypothesis. Thus, what is considered part of the model in one analysis can become part of the hypothesis in another. In the same way, something considered part of the hypothesis in one type of analysis can become part of the model in another, depending on the question being asked. An example of this is Pagel’s (1994) approach to measuring and testing correlations among traits. This approach treats the topology (and branch lengths) of the tree as fixed, the question being ”Is there a significant association between traits given this particular topology having these particular branch lengths.”

15.3 The principle of maximum likelihood

The likelihood provides a way to quantify our belief in a particular hypothesis. The principle of maximum likelihood states that, presented with two competing hypotheses, our best guess (inference) at the truth involves choosing the hypothesis having the higher likelihood. This is tantamount to saying (assuming our model is true) that we should choose as best the hypothesis that makes the observed data most probable. This principle is very generally applicable, and thus there is no such thing as the maximum likelihood method. The formula for computing the likelihood depends entirely on the model and the question being asked.

The generality of likelihood means that it is a very flexible method. Thus, if it is determined that some aspect of the model being used is flawed, once has only to reformulate the model, taking into account this new factor, and redo the analysis. For example, the effect of among-site rate variation on phylogenetic inference has now become widely appreciated (Huelsenbeck, 1995a). Many phylogeny reconstruction methods fail if their underlying assumption of equal rates among sites is violated. A method relying on the principle of maximum likelihood will also fail if the model being used does not take into account the possibility of rate heterogeneity (Chang, 1996; Gaut and Lewis, 1995; Huelsenbeck, 1995a); however, it is possible to take rate heterogeneity into account in a likelihood model and hence remove it as a problem (Felsenstein and Churchill, 1996; Yang, 1993, 1994b). While skepticism is a natural response when faced with the many assumptions necessary in order to compute the likelihood, the explicit enumeration of assumptions is useful in that it allows us to recognize the aspects of our model most in need of scrutiny. Perhaps the biggest challenge of the coming decade lies in identifying those aspects of our evolutionary models that are most suspect, and revising our current suite of models to accomodate these factors. Not all known aspects of sequence evolution are embodied in every data set. For example, pseudogenes often do not show marked variation in substitution rates across sites, whereas protein-coding genes exhibit much higher rates at third-codon (silent) positions than at first- or second-position sites. It may be possible to use a relatively simple model that nevertheless accounts for the most important factors in the evolution of the particular gene and group under study (reference). Likelihoods thus provide not only a way to choose among tree topologies, but also a way to evaluate the adequacy of the model itself.

15.4 Maximum likelihood estimation

Likelihoods can be used to choose among competing tree topologies in a manner analogous to the way we chose among different die models, above. This is because different tree topologies makes different predictions about the frequency of site patterns in our sequence data. Furthermore, likelihood can be used to estimate the branch lengths within a single tree topology. This is because two trees having the same topology but different branch lengths also constitute different hypotheses. The following example will illustrate that branch lengths are central in the calculation of the likelihood of a tree. We will use for our example the simplest possible tree: one branch connecting two taxa:

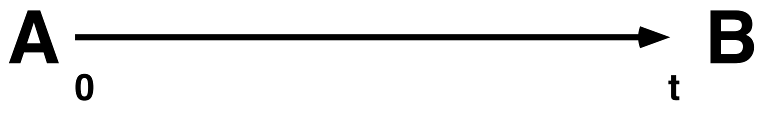


Figure 15.1 *a single branch.*

For this example, let's assume taxon A is the ancestor of taxon B, and the length of the branch will be the expected number of changes per site that have occurred over the time (t) that the lineage has been evolving. The data we have observed are counts of the number of differences between taxa A and B. Let's say that out of 100 two-state characters, we observed 10 to be different in state between A and B. If our characters change state at a rate μ changes per site per year, then the branch length (expected number of changes per site) will be νt. Let ν be the product of the rate of change and time. How does the branch length (ν) affect the probability of observing 10 characters out of 100 to differ? We need a model that relates branch length to the probability of observing a difference between two taxa separated by t years when the rate of substitution has been a constant μ. The formula[*], often called the Cavender-Farris model (references), is given as:

$$Pr(diff;\nu) \quad = \quad \frac{1}{2}(1-e^{-2\nu}) \tag{15.1}$$

where Pr(diff; ν) means "the probability of a difference (between A and B) given branch length ν." The table below shows how the probability of observing a difference at the ends of the branch depends on the branch's length (ν):

Effect of increasing branch length on the probability of observing a difference

ν	Pr(diff;ν)
0.01	0.010
0.05	0.048
0.10	0.091
0.5	0.316

It is clear that as the branch length increases, so does the probability of observing a difference across the branch. It is important to note again that

[*]The reasoning that allows us to obtain formulas such as the one below will be explained under the topic "Transition Equations."

the branch length n is the product of the rate of change per site and time. This means that a branch of a particular length could result from a high rate of change over a short period of time *or* a slow rate of change over a longer period of time. We seldom have the information necessary to disentangle rates from times, but fortunately we never need to separate the rate from the time because it is only the product (v) of these two quantities that enters into the formulas we use to compute probabilities.

15.4.1 The likelihood curve

Suppose now that we have a dial attached to the branch that we can turn. If we turn it in a clockwise direction, the branch lengthens; if we turn it in a counter-clockwise direction the branch shortens. As we turn the dial, a gauge shows us the probability of observing our data given the current length of the branch. Let's start by turning the dial counter-clockwise until the branch has zero length. Our gauge tells us that the probability of observing 10 differences out of 100 characters is zero when the branch length is zero. This makes perfect sense: a branch length of zero indicates that either 1) the substitution rate is zero, or 2) the time that the lineage has been evolving is zero. In both of these possible scenarios, there is no opportunity for a change to take place, and thus no chance of the ten (or more) substitutions needed to produce the observed data. It also should make sense that if we turn the dial in a clockwise direction long enough, we will end up with a branch so long that we should expect *many more* than 10 differences between our two taxa. If we were to plot the values on our gauge as we turned the dial, we would produce a graph that looked like the one in the figure below.

Note that the probability of observing our data is greatest when the length of the branch is about 0.11 (see the dotted vertical line). If the branch length is greater than 0.11, one would expect more change than was observed. For branch lengths less than 0.11, less change than that observed would be expected. The value 0.11 is called the ***maximum likelihood estimate*** of the branch length parameter v. When the parameter v is set to this value, the probability of observing 10 differences out of 100 sites is as high as it can be.

15.4.2 Likelihoods and probabilities

You might be wondering at this point how the plot above was calculated. The probability of the data (i.e., the likelihood) for this example is a

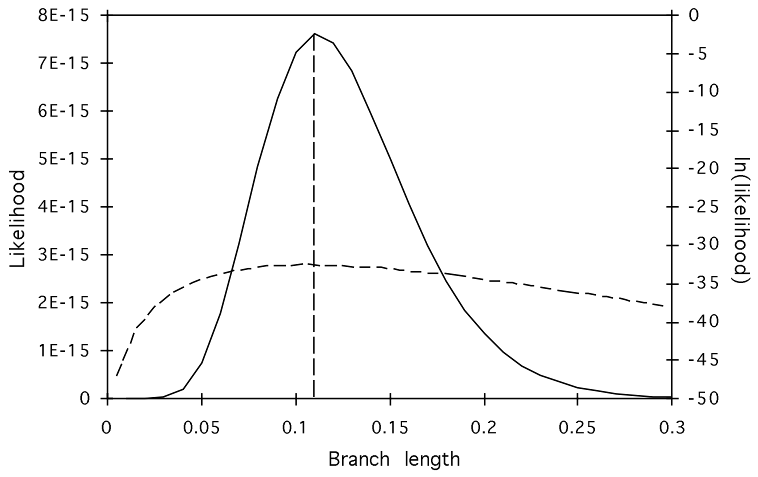


Figure 15.2 *Plot of the likelihood for branch lengths ranging from 0.0 to 0.3. Note that the likelihood function reaches its maximum at the branch length 0.11.*

simple function of the probability of a difference given a specific value for v:

$$Pr(data; v) = [Pr(diff; v)]^{10}[1 - Pr(diff; v)]^{90} \tag{15.2}$$

The probability of observing a difference is raised to the power 10 because 10 differences were observed, and the probability of 10 independent events is just the product of the probabilities of each event. Likewise, the probability of observing no difference at a character is just one minus the probability of a difference, and this quantity must be raised to the power 90 because 90 characters were observed to have identical states in both taxon A and taxon B. To obtain a numerical value for the likelihood, we must select a value to plug in for the branch length, v, in the likelihood equation. Selecting a slightly different value for v means you have a slightly different hypothesis and the likelihood will assume a different numerical value. Seeking the maximum likelihood estimate of v simply involves successively comparing hypotheses differing only in the value of this one parameter, and choosing that value that yields the highest likelihood.

15.4.3 *Properties of maximum likelihood estimators*

The value 0.11 obtained above for n is called the maximum likelihood estimate, or MLE, of v. Maximum likelihood estimators have several desirable statistical properties, most notably statistical consistency and efficiency. Statistical consistency means that as more data are added, the closer the MLE gets (on average) to the parametric ("true") value. An inconsistent estimator will converge on the *wrong* value as larger and larger data sets are considered! Thus, statistical consistency is a very desirable feature in an estimator. The efficiency of an estimator measures its variance against the theoretical lower bound of the variance. In other words, an estimator that is less efficient than another requires more data in order to achieve the same accuracy. Under fairly general conditions, it can be shown that maximum likelihood estimators are asymptotically normally distributed and have a variance equal to this theoretical lower bound. This means that maximum likelihood estimators are among the most efficient estimators available. The utility of maximum likelihood estimation for the estimation of phylogeny has been demonstrated using computer simulation (Huelsenbeck, 1995b,a; Gaut and Lewis, 1995). Estimators can also be compared on the basis of bias. Bias occurs when the expected value of an estimate is not equal to the parametric value. Maximum likelihood estimators are often biased, but the amount of bias is minimal if the data set is of reasonable size. The relationship between bias and variance can be easily visualized by thinking of a target sport, such as archery. An archer whose arrows always land below the bullseye (biased) but which are never more than 1 inch from the bullseye (low variance) would be preferred over one whose aim is unbiased but whose arrows usually land more than a foot away from the bulleye.

15.4.4 *Likelihoods of trees*

A new issue must be confronted in order to make the move to calculating likelihoods for trees having more than one branch: the problem of what to do about the states at interior nodes of the tree. Consider the three-taxon rooted tree below:

The three tips of this tree are labeled with the states of the three taxa at one nucleotide site and the question marks indicate that we have no way of knowing the states for this site at the two interior nodes. There are two ways to handle this situation: 1) we could estimate the ancestral states, calculating the likelihood based on our best guess for these nodes, or 2) we could take all 16 possible combinations of nucleotides into

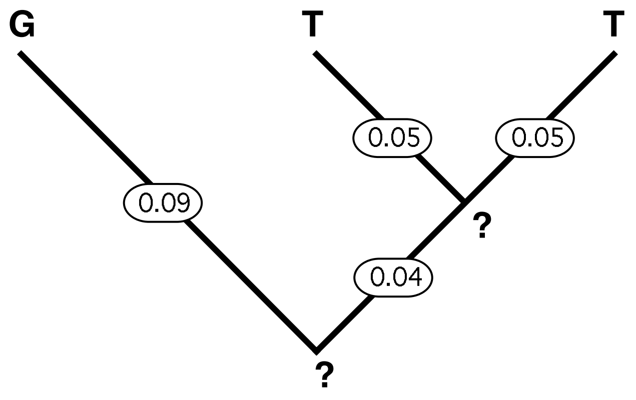


Figure 15.3 *three-taxon tree.*

consideration when computing the likelihood. The likelihood computed under the first option is only as good as our estimates of the unknown states. The second option allows us to get around this problem, but requires several times the computational effort of the first option.

No major likelihood program uses the first approach, the reason being that, by estimating these states at the interior nodes, we add many more parameters to our model. For the tree above, we would have two additional parameters needing to be estimated for *each site*! What's worse, we have only the data for one site with which to estimate these parameters; estimates of branch lengths make use of data from all sites. Using so little data for estimation is risky, and can lead to statistical inconsistency (Felsenstein, 1978a, 1983; Goldman, 1993).

The second option requires that we consider every possible state at each of the interior nodes. Even though this requires a lot of computation, it means that the optimality criterion we are using (the likelihood of the tree) makes no assumptions about the unknown states at these ancestral nodes. The table below shows the likelihoods obtained by assuming, in turn, each of the sixteen possible combinations of states at the two interior nodes of the tree above:

The sixteen possible combinations of states at the interior nodes of the example tree above and the components of the likelihood associated with each assuming a Jukes-Cantor model of nucleotide substitution.

States	Likelihood
AA	0.00066
AC	0.00039
AG	0.00039
AT	0.00138
CA	0.00039
CC	0.00066
CG	0.00039
CT	0.00138
GA	0.00106
GC	0.00106
GG	0.00180
GT	0.00378
TA	0.00039
TC	0.00039
TG	0.00039
TT	0.00234
Total	0.01687

The substitution model used here is that of Jukes and Cantor (1969), which is explained in more detail in the section ″Transition Equations.″ This model assumes that all nucleotides occur with equal frequencies and, when a substitution occurs, the substituted base can be any base except the one present before the substitution occurred. The overall likelihood is obtained by adding up all sixteen individual likelihoods (we add because the truth could be any one of these 16 mutually-exclusive possibilities: see ″Appendix A: History of ML phylogeny inference″). Note that the ″most likely″ combination of states at the interior nodes is GT, with a likelihood of 0.00378, but other combinations (e.g., TT) contribute a substantial amount to the overall likelihood (0.01687). Note also that GT and TT are both most parsimonious reconstructions (each requires only one step on the rooted tree). but they do not result in the same likelihood: GT (0.00378) is slightly preferred over TT (0.00234).

How are these 16 likelihoods calculated? Each is the product of five terms, with four of these terms corresponding to the four branches of the tree. The fifth term is the frequency of the base at the root node (0.25 for the Jukes-Cantor model). For the case in which both interior nodes have base A, the event of interest can be phrased as follows:

we want the probability of ″starting with base A at the root node

AND changing from A to G across a branch of length 0.09
AND remaining the same (i.e., staying base A) across a branch of length 0.04

AND changing from A to T across a branch of length 0.05
AND changing from A to T across a branch of length 0.05".

This sample point represents an intersection of five separate subevents, and the probabilities of each of these events must therefore be multiplied together to get the probability of the pattern GTT (given A at both interior nodes). The probability of starting with base A at the root involves the frequency of base A (πA), and the other four terms are ***transition probabilities*** obtained from plugging the appropriate branch lengths into the appropriate ***transition equation***, which is a formula derived from the substitution model being used. A description of how transition equations are derived is given later (see "Transition Equations").

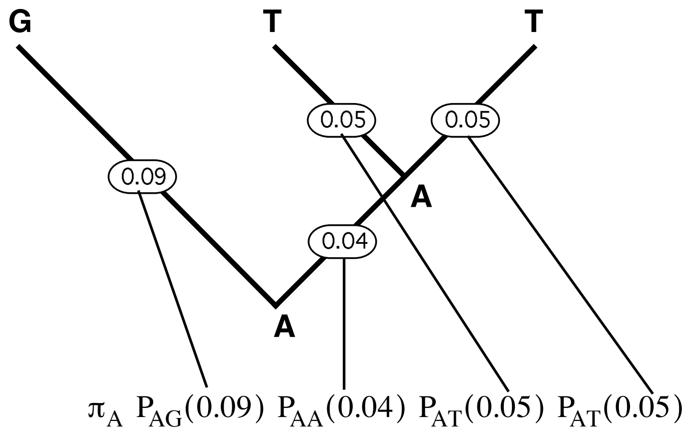

G
T
T
0.05
0.05
0.09
A
0.04
A
π_A $P_{AG}(0.09)$ $P_{AA}(0.04)$ $P_{AT}(0.05)$ $P_{AT}(0.05)$

Figure 15.4 *Three taxon tree showing the states G, T, and T at the tip nodes and A at both the root node and the other interior node. The probability of the pattern GTT (given the states at the interior nodes) is shown below the tree.*

For the Jukes-Cantor model, the equation for a transition to any non-identical base is

(15.3) $$P_{ij}(v) \quad = \quad 0.25(1 - e^{-4v}$$

whereas the probability of no change over a branch of length n is

$$P_{ij}(v) \quad = \quad 0.25 + 0.75e^{-4v} \tag{15.4}$$

For this example, the frequency of base A is just 0.25 and the four transition probabilities are:

$$P_{AG}(0.09) = 0.25(1 - e^{-4(0.09)} = 0.17442$$
$$P_{AA}(0.04) = 0.25 + 0.75e^{-4(0.04)} = 0.36089$$
$$P_{AT}(0.05) = 0.25(1 - e^{-4(0.05)} = 0.20468$$
$$P_{AT}(0.05) = 0.25(1 - e^{-4(0.05)} = 0.20468$$

The product of these five terms is 0.00066, in accord with the first row of the table above. To get the likelihood of the tree then, we begin with an initial guess at the four branch lengths and compute the likelihood (summing over all 16 possibilities at the interior nodes). Then we adjust the branch lengths, recalculating the likelihood after each adjustment, until we can no longer improve the likelihood by adjusting any branch length by any amount. If we were forced to use a trial-and-error approach to such adjustment of branch lengths, this process would be very slow indeed; however, such a situation is encountered in many different kinds of problems and numerical methods have been developed that allow us to make educated guesses about the direction and magnitude of each adjustment so that we can quickly obtain maximum likelihood estimates of all the branch lengths in a tree (for more on the specific method employed by PAUP*, see "Estimation of branch length parameters").

Dealing with data from many sites

The likelihood above used data from only one nucleotide site. How do we calculate a likelihood when our data comprises many nucleotide sites? Let's call the likelihood above L_i (the likelihood for site i). What we really need is

$$L \quad = \quad L_1 L_2 L_3 ... L_N \tag{15.5}$$

where N is the total number of sites in our sequence alignment. The data for each site represents the outcome of an experiment, the word experiment being used here in its statistical sense meaning a single run of a stochastic process (not necessarily under the control of the "experimenter"). The "experiment" here is the evolution of a single nucleotide site over time. Because of the stochastic nature of the substitution process, we would probably not get the same outcome for any particular site were we able to do it all over again, even if we started with the same ancestral base. Since we are interested in the joint outcome of N such evolutionary experiments, we must multiply the likelihoods for individual sites to obtain the probability of "the pattern at site 1 AND the pattern at site 2 AND ... AND the pattern at site N". Note that this involves the assumption that the evolution of the pattern at one site is independent of the evolution of the patterns at all other sites. This is not really a good assumption to have to make, but it is made by nearly all methods for phylogenetic inference, including parsimony and distance methods. Models have been proposed that allow for some dependence among sites, such as the dependence between DNA sites comprising a single codon (Goldman and Yang, 1994; Muse and Gaut, 1994) and the dependence between internally paired sites in ribosomal RNA (Muse, 1995; Tillier and Collins, 1994), but taking account of this dependence comes at the cost of additional computational effort. Tillier and Collins (1994) found that maximum likelihood methods were surprisingly robust to violations of this key model assumption in the case of ribosomal stem site dependencies.

15.4.5 Combining site patterns

We can save considerable computation by noting that often several sites show the same pattern of nucleotides at the tip nodes of the tree. This is especially true for the constant sites; patterns such as AAA, CCC, GGG, and TTT for our three-taxon tree. Since the data are the same for such sites having common patterns, their likelihoods will be the same, so we can avoid much unnecessary calculation by simply packing the data matrix before we start, keeping track of the number of sites sharing each pattern. The data matrix now consists of M distinct patterns, each associated with a count of the number of sites having that pattern:

$$L = L_1^{c_1} L_2^{c_2} L_3^{c_3} ... L_N^{C_M} \qquad (15.6)$$

15.4.6 *Advantages of using log-likelihoods*

Now we are faced with another problem. Likelihoods computed for real sequence data involving perhaps a thousand or more nucleotide sites end up being numbers on the order of 10^{-500} or so. Why is this a problem? In comparing numbers of this magnitude, round off errors can cause considerable problems. Round off errors occur because only a certain number of digits of precision can be maintained in a computer's memory; the remainder are ignored. Fortunately, there is an easy solution which not only solves the round off error problem but also allows us to compare numbers that don't have 500 zeros in front of them! The solution is to compare logarithms of likelihoods rather than the likelihoods themselves. The logarithm of a product of two numbers equals the sum of the logarithms of each term (see box below). This allows us to deal with a sum rather than a product,

$$\ln(L) = c_1 \ln(L_1) + c_2 \ln(L_2) + c_3 \ln(L_3) + \ldots + c_M \ln(L_N) \tag{15.7}$$

Laws of natural logarithms

1. The natural logarithm is the inverse of the exponential function:

$$a = e^{\ln(a)} = \ln(e^a)$$

2. The log of a product of two terms is the sum of the log of each term:

$$\ln(ab) = \ln(a) + \ln(b)$$

3. The log of a number raised to a power is the power times the log of the number:

$$\ln(a^b) = b \ln(a)$$

Products are much more susceptible to round off problems than sums because in products small errors are compounded. Also fortunate is the fact that once you find a combination of parameter settings that maximizes the ln-likelihood function, the same set of parameter settings also maximizes the likelihood function, as can be seen in the graph below (which is the same plot as the one shown before (need cross ref) for the single branch example).

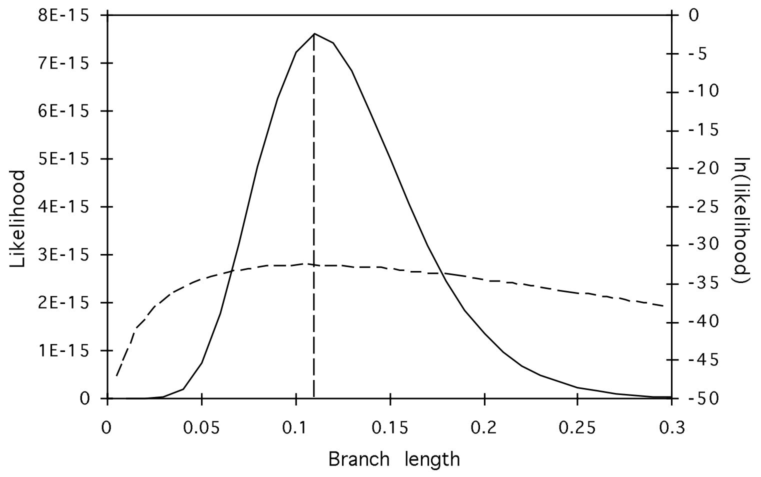


Figure 15.5 *Likelihood (solid line, left vertical axis) and ln-likelihood (dotted line, right vertical axis) functions for the length of a branch (same data as that used for previous plot). Note that both functions reach their maximum at the same branch length (about 0.11).*

In summary, calculation of the likelihood for a particular tree topology for comparison with other tree topologies requires the following steps:

1. Select starting branch lengths for all branches in the tree. These can be wild guesses, or better, estimated from a matrix of pairwise distances.
2. Compute the likelihood of the tree over all sites using equation (4.7).
3. Adjust the branch lengths slightly (using some method that suggests intelligent choices for the direction and magnitude of each adjustment), and recalculate the likelihood using the adjusted branch lengths.
4. Repeat step 3 until the likelihood changes less than a certain specified amount from one iteration to the next.

15.4.7 *Taking account of site-to-site rate heterogeneity*

Thus far we've assumed that there is a single rate of evolution that applies equally to all sites in the sequence. We never have to estimate this quantity since it is confounded with the time that a lineage has been in existence. The fact that time and rate always appear as a product in the likelihood formulations (this product is what we've been referring to as the branch length) also means that the rate of evolution can be different for every branch of the tree. We still must assume, however, that whatever rate applies to branch b for one site is the same as the rate for that branch at all other sites. We know that this is a particularly bad assumption to have to make, since rates are known to vary among sites. For example, the third codon position in protein-coding gene sequences is usually much more variable among taxa than are first and second codon positions. Fortunately, this site-to-site rate heterogeneity is not difficult to take account of in models of sequence evolution. The approach used in PAUP* assumes that the distribution of rates across sites can be approximated by a gamma (Γ)distribution (Yang, 1993). The continuous gamma distribution is fully described by two parameters, called the shape and the scale:

$$f(r) = \frac{r^{\alpha-1}e^{-r/\beta}}{\beta^{\alpha}\Gamma(\alpha)} \tag{15.8}$$

The α in the above gamma density function is the shape parameter, and the β is the scale. The r represents the relative rate factor and the Γ is called the gamma function. The absolute rate is the relative rate factor, r, times the base substitution rate. The gamma distribution has a mean of $\alpha\beta$. For our purposes, it is convenient to set the scale equal to the inverse of the shape (i.e., $\beta = 1/\alpha$), resulting in a gamma distribution always having mean 1.0 (which is a necessary feature for a distribution of relative rates). We can now vary the distribution to accommodate a wide range of possible types of rate heterogeneity simply by changing the shape parameter alone. The following figure illustrates the continuous gamma distribution (with $\beta = 1/\alpha$) for various values of α:

When the shape parameter is very large, say 50, the gamma distribution consists of a sharp peak at the value 1.0, indicating very little heterogeneity among sites. If one were to randomly draw values from this distribution, most would fall very close to 1.0, and thus the actual

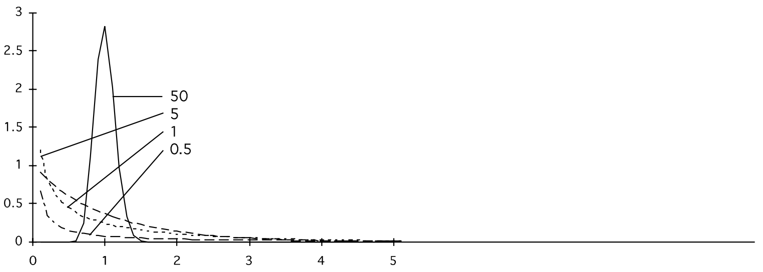


Figure 15.6 *The gamma distribution, with scale equal to the inverse of the shape, for four different values of the shape parameter. Note that for high values of the shape parameter, the distribution becomes a sharp peak centered at 1.0, corresponding to near homogeneity of rates across sites, whereas at low values the distribution becomes quite skewed.*

rates would all be very close to the base rate. For values of the shape parameter much less than 1.0, the gamma distribution predicts many relative rates near 0.0, but some relative rates that are very large. Nearly the whole gamut of rate heterogeneity can thus be summarized by this one parameter.

In order to make this approach tractable, PAUP* uses a discrete approximation to the gamma distribution (described by Yang, 1994b). The user must specify the number of rate categories to use in the model and the program then selects heights and relative rates for each of these categories such that a gamma distribution with a given shape is approximated. To accomplish this, we first determine k-1 cutoff points (for k categories) such that the area under the gamma distribution is $1/k$ for each interval between cutoffs. For example, if there were 4 categories, we would find points along the x-axis such that 25% of the area of the gamma distribution was contained between each pair of adjacent points. Once these cutoff points are determined, we can get the relative rate for each category by either taking the mean or the median of each section (PAUP* allows either the mean or the median to be used for finding relative rates). The likelihood then considers, for each site, the possibility that the site has each of these k relative rates. This is similar to the way it takes into account all four possible bases at ancestral nodes: the likelihood for one site can be written as follows

$$L = \sum_{j=1}^{k} \Pr(\text{data for site given rate} = r_j)\Pr(\text{rate} = r_j)$$

The term Pr(rate = r_j) is just $1/k$ for each rate (since we made each category cut off an equal amount of the distribution), and the term Pr(data for site given rate = r_j) is just the likelihood computed by substituting the product vr_j in for n in all the transition equations involved. By "integrating out" the relative rates in this way, we are effectively adding only one additional parameter to the model, the gamma shape parameter. PAUP* allows this parameter to be estimated for a given data set, although this is a time consuming option. Estimates of the shape parameter do not appear to change much from topology to topology (Yang, 1994b), and thus it is probably nearly as good to obtain an estimate from one topology and simply fix the shape parameter at this value for the duration of a large search.

The next section discusses substitution models determining the form of the transition equations used to compute the probabilities of changes across each branch of the tree.

15.5 Substitution models

At the center of any maximum likelihood method for inferring phylogeny, whether employing nucleotide sequence data (Felsenstein, 1981a; Hasegawa et al., 1985), restriction site data (Felsenstein, 1992), gene frequencies (Felsenstein, 1973), or morphological data (Felsenstein, 1973; Pagel, 1994; Lewis, 2001), is an evolutionary model that turns relative rates of substitution and divergence times into probabilities of specific types of character change. In the first section, we showed how the likelihood for a single site consists of a product involving the frequency of the base at the root node and transition equations for each branch of the tree. These transition equations describe the mathematical consequences of the application, over large spans of time ,of probabilities defined for an infinitesimal period of time.

If we were to plot the probability of a specific transition, say the change $A \rightarrow C$, over time, it would look something like this:

Note that the probability of this change is nonlinear and is, like all probabilities, always less than or equal to 1.0. Instead of continuing to rise over time, this probability eventually levels out at a value determined by the substitution model. This value is the stationary frequency of the

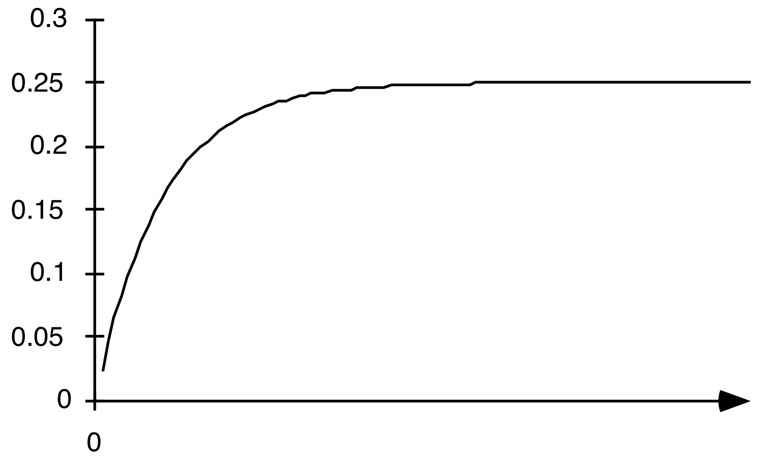


Figure 15.7 *Plot of PAC(t) (vertical axis) against time (horizontal axis).*

base being substituted to. This is because after a very long time, many substitutions have occurred and the probability of seeing any particular base is the same as the probability of simply choosing that base at random. If we wait long enough, all information about time will be lost because of the randomizing effect of multiple hits. Because of this nonlinearity, it is not a trivial problem to predict the transition probability for an arbitrary time *t* given the rate of change during an instant of time

This section describes how this is accomplished, making use of several results from the field of mathematics known as stochastic processes. We will begin with one of the most general substitution models possible, and show how all of the models implemented in PAUP* are simply special cases of this very general model.

Before we dissect one of these terms into its component parts, let's take a moment to consider what we mean by an instantaneous rate. The transition equation plotted above results from the Jukes and Cantor (1969) model of substitution. This model will be described in detail below, but for now all we want to do is show the relationship between the transition probability plotted in the graph and its corresponding instantaneous rate. The equation for the transition probability is

$$P_{AC}(t) \quad = \quad 0.25(1 - e^{-\mu t}) \tag{15.9}$$

where *e* is the base of the natural logarithms (having a numerical value approximately 2.718), m is the substitution rate, and t is the elapsed time. For large amounts of time, say $t = \infty$, $e^{-\infty} = 1/e^{\infty}$, making $PAC(t) = 0.25$. After no time has elapsed, $t = 0, e^{-\mu t} = e^0 = 1$ and $PAC(t) = 0.0$, as in the plot. Even though the probability of the change $A \rightarrow C$ is zero at time $t = 0$, this probability does grow as time elapses, so there must be some positive rate associated with $PAC(t)$, even at $t = 0$. This rate is the instantaneous rate and we can find out what it is by finding the slope of the curve at $t = 0$. The slope of this curve at $t = 0$ is computed by finding the derivative of the transition equation, and then plugging in 0 for *t* in the resulting formula. Since, for an arbitrary *x*, the derivative of *ext* with respect to *t* is just *ext* times the derivative of *xt* with respect to *t*, the derivative of $PAC(t)$ with respect to *t* is

$$\frac{\partial}{\partial t}P_{AC}(t) \quad = \quad 0.0 - 0.25e^{-\mu t}(-\mu) = 0.25\mu e^{-\mu t} \tag{15.10}$$

and this equation evaluated at $t = 0$ yields 0.25μ. Thus, the rate of change after only an infinitesimal amount of time has elapsed is one fourth of the substitution rate for the Jukes and Cantor (1969) model of substitution. Our goal in the remainder of this section is to show how to go the other way; that is, starting with the instantaneous rate, how do we obtain an expression for the transition probability after an arbitrary amount of time has elapsed.

We begin by specifying these ***instantaneous rates*** for all possible changes among the states. A matrix with four rows and four columns provides a convenient way to represent these rates for nucleotide sequence data:

$$\mathbf{Q} \quad = \quad \begin{bmatrix} --- & \mu a\pi_C & \mu b\pi_G & \mu c\pi_T \\ \mu g\pi_A & --- & \mu d\pi_G & \mu e\pi_T \\ \mu h\pi_A & \mu i\pi_C & --- & \mu f\pi_T \\ \mu j\pi_A & \mu k\pi_C & \mu l\pi_G & --- \end{bmatrix} \tag{15.11}$$

The rows and columns of this matrix correspond, respectively, to the nucleotides A, C, G, and T. An element at row i, column j of this matrix (i.e., $\mathbf{Q}_{ij}$) corresponds to the rate of a change from base i to base j during an infinitesimal time period which we will hereafter refer to as dt. The diagonal elements of this matrix have dashes because the values in these cells are completely determined by the remaining elements on the same row. For example, since an A at a particular site must either remain the same or change to one of the other three bases during the time dt, the four rates making up the first row must add up to 0.0. That is, if there is some "leakage" over time of As to other bases, this must necessarily entail a loss in the number of remaining As. This translates to a negative rate for the transition $A \rightarrow A$ that is equal in magnitude to the sum of the (positive) rates for the transitions $A \rightarrow C$, $A \rightarrow G$, and $A \rightarrow T$.

Let's take a single element, say $\mathbf{Q}_{AC}$, and examine each of its component terms in order to understand why this represents the rate of change from an A to a C during the tiny period of time dt. The element $\mathbf{Q}_{AC}$ is $\mu\alpha\pi C$. The term μ will be referred to as the ***mean instantaneous substitution rate***. The term α provides the relative rate of a change from A to C, and will be termed a ***rate parameter***. There are twelve such rate parameters, designated $a, b, c, ..., l$, one for each possible transition from one base to a non-identical base. The rate parameters always form a product with the mean substitution rate μ, and simply act to modify μ. The frequency parameter πC allows us to modify the mean instantaneous rate to take account of the possibility that the bases do not all have the same frequency. Thus, for an extreme example, we might not want the instantaneous rate for the change to be greater than zero if, for some reason, there were no Cs (i.e., $\pi C = 0.0$) in the pool of nucleotides available at the moment the substitution event occurred. Including the frequency of the base being substituted into the expression for the instantaneous rates allows us to take account of heterogeneity of base frequencies without suffering any cost if the bases really are all equal in frequency.

The preceding discussion will no doubt become clearer if we show how the general model represented by (-3-3) collapses to the simpler models employed by PAUP*. The simplest model available is the ***Jukes & Cantor (1969) model***, in which only one class of substitutions are defined and the nucleotides are assumed to be in equal frequencies. Thus,
$\pi A = \pi C = \pi G = \pi T = 0.25$ and $a = b = c = ... = l = 1.0$, reducing the infinitesimal rate matrix $\mathbf{Q}$ to

(15.12)
$$\mathbf{Q} = \begin{bmatrix} -\frac{3}{4}\mu & \frac{1}{4}\mu & \frac{1}{4}\mu & \frac{1}{4}\mu \\ \frac{1}{4}\mu & -\frac{3}{4}\mu & \frac{1}{4}\mu & \frac{1}{4}\mu \\ \frac{1}{4}\mu & \frac{1}{4}\mu & -\frac{3}{4}\mu & \frac{1}{4}\mu \\ \frac{1}{4}\mu & \frac{1}{4}\mu & \frac{1}{4}\mu & -\frac{3}{4}\mu \end{bmatrix}$$

In most treatments of this model, the frequency (1/4) and the substitution rate (μ) are combined into a single parameter (call it α), resulting in the (slightly) more appealing form

$$\mathbf{Q} = \begin{bmatrix} -3\alpha & \alpha & \alpha & \alpha \\ \alpha & -3\alpha & \alpha & \alpha \\ \alpha & \alpha & -3\alpha & \alpha \\ \alpha & \alpha & \alpha & -3\alpha \end{bmatrix}$$

The ***Kimura (1980) model*** is slightly less restrictive in allowing two classes of substitutions (transition-type and transversion-type), but the base frequencies are still assumed to be equifrequent. Thus, we force $a = c = d = f = g = i = j = l$ all to equal 1 and $b = e = h = k$ all to equal the value κ, where κ represents the ratio of transition- to transversion-type substitution rates. Again, the base frequencies $\pi A = \pi C = \pi G = \pi T$ are all set to 1/4, yielding

$$\mathbf{Q} = \begin{bmatrix} -\frac{1}{4}\mu(\kappa+2) & \frac{1}{4}\mu & \frac{1}{4}\mu\kappa & \frac{1}{4}\mu \\ \frac{1}{4}\mu & -\frac{1}{4}\mu(\kappa+2) & \frac{1}{4}\mu & \frac{1}{4}\mu\kappa \\ \frac{1}{4}\mu\kappa & \frac{1}{4}\mu & -\frac{1}{4}\mu(\kappa+2) & \frac{1}{4}\mu \\ \frac{1}{4}\mu & \frac{1}{4}\mu\kappa & \frac{1}{4}\mu & -\frac{1}{4}\mu(\kappa+2) \end{bmatrix}$$

Once again, the form of this matrix most often presented involves some simplification, with $\alpha = \frac{1}{4}\mu$ being called the transition rate and $\beta = \frac{1}{4}\mu\kappa$ the transversion rate. This results in

$$\mathbf{Q} = \begin{bmatrix} -(\alpha+2\beta) & \beta & \alpha & \beta \\ \beta & -(\alpha+2\beta) & \beta & \alpha \\ \alpha & \beta & -(\alpha+2\beta) & \beta \\ \beta & \alpha & \beta & -(\alpha+2\beta) \end{bmatrix}$$

The ***Hasegawa, Kishino, Yano (1985) model*** adds one more level of flexibility to the Kimura (1980) model by allowing unequal base frequencies:

$$\mathbf{Q} = \begin{bmatrix} -\alpha\pi_G - 2\beta\pi_Y & \beta\pi_C & \alpha\pi_G & \beta\pi_T \\ \beta\pi_A & -\alpha\pi_T - \beta\pi_R & \beta\pi_G & \alpha\pi_T \\ \alpha\pi_A & \beta\pi_C & -\alpha\pi_A - \beta\pi_Y & \beta\pi_T \\ \beta\pi_A & \alpha\pi_C & \beta\pi_G & -\alpha\pi_C - \beta\pi_R \end{bmatrix}$$

where $\alpha = \mu\kappa, \beta = \mu, \pi_R = \pi_A + \pi_G, and \pi_Y = \pi_C + \pi_T$.

The ***Felsenstein (1981) model*** is identical to the Jukes and Cantor (1969) model except for allowing heterogeneity in base frequencies. Thus, the relative rate parameters $(a, b, c, ..., l)$ are all set to 1.0 but the frequency parameters remain untouched from the general model:

$$\mathbf{Q} = \begin{bmatrix} -\mu(\pi_G + \beta\pi_Y) & \mu\pi_C & \mu\pi_G & \mu\pi_T \\ \mu\pi_A & -\mu(\pi_T + \mu\pi_R) & \mu\pi_G & \mu\pi_T \\ \mu\pi_A & \mu\pi_C & -\mu(\pi_A + \mu\pi_Y) & \mu\pi_T \\ \mu\pi_A & \mu\pi_C & \mu\pi_G & -\mu(\pi_C + \mu\pi_R) \end{bmatrix}$$

The *Felsenstein (1984) model* allows for both transition- and transversion-type substitutions, but models this transition:transversion bias differently than either the Kimura (1980) model or the Hasegawa et al. (1985) model. YYY Instead of thinking in terms of a transition rate and a transversion rate, this model requires that we think in terms of a *general* substitution rate, which produces all types of substitutions, and a special *within-group* rate, which produces only transition-type substitutions and hence (if non-zero) serves to enrich the rate of transition-type substitutions over the transversion-type substitutions. To fit this into our general model, we set $a = c = d = f = g = i = j = l$ to 1, as we did for the K2P and HKY models, and:

$$\mathbf{Q} = \begin{bmatrix} --- & \mu\pi_C & \mu\pi_G(1+\frac{\kappa}{\pi_R}) & \mu\pi_T \\ \mu\pi_A & --- & \mu\pi_G & \mu\pi_T(1+\frac{\kappa}{\pi_Y}) \\ \mu\pi_A(1+\frac{\kappa}{\pi_R}) & \mu\pi_C & --- & \mu\pi_T \\ \mu\pi_A & \mu\pi_C(1+\frac{\kappa}{\pi_C}) & \mu\pi_G & --- \end{bmatrix}$$

The elements for transition-type substitutions involve two terms because they can result from either a general substitution event or a within-group substitution event. The frequency of the base being substituted is simply π_j in the case of a general substitution, where j is the substituted base. In the case of a within-group substitution, however, the frequency used is the frequency of base j (the substituted base) divided by the frequency of the group (purine or pyrimidine) to which base j belongs. Although it is not apparent, we are doing the same thing for the general substitutions. The "group" in the case of a general substitution comprises all four nucleotides and thus the frequency used could be written as

$$\frac{\pi_j}{\pi_A+\pi_C+\pi_G+\pi_T}$$

but is usually not since the denominator simplifies to 1.

The last model is the most general model offered by PAUP*. This ***general 6-parameter model*** differs from the one in equation (-3-3) only in that there are six classes of substitution instead of twelve: that is, $g = a, h = b, i = c, j = d, k = e$, and $l = f$:

$$\mathbf{Q} = \begin{bmatrix} --- & \mu a\pi_C & \mu b\pi_G & \mu c\pi_T \\ \mu a\pi_A & --- & \mu d\pi_G & \mu e\pi_T \\ \mu b\pi_A & \mu d\pi_C & --- & \mu f\pi_T \\ \mu c\pi_A & \mu e\pi_C & \mu f\pi_G & --- \end{bmatrix}$$

This model allows there to be six different classes of substitution possible. The other five models force substitutions into either one (F81 and JC) or two (K2P, HKY, and F84) classes.

15.5.1 *From rates to probabilities*

The instantaneous rates represent the rate of a particular transition per instant of time dt, and thus multiplying by dt provides the probability of this specific change over the time period dt. For example, $\mu a \pi_C dt$ is the infinitesimal probability $P_{AC}(dt)$ of the transition $A \rightarrow C$. We can represent all the infinitesimal probabilities in matrix form as follows:

$$\mathbf{P}dt = \begin{bmatrix} --- & \mu a \pi_C dt & \mu b \pi_G dt & \mu c \pi_T dt \\ \mu g \pi_A dt & --- & \mu d \pi_G dt & \mu e \pi_T dt \\ \mu h \pi_A dt & \mu j \pi_C dt & --- & \mu f \pi_T dt \\ \mu i \pi_A dt & \mu k \pi_C dt & \mu l \pi_G dt & --- \end{bmatrix} \tag{15.13}$$

The diagonal elements now equal to *one* minus the other elements on the same row, because each matrix element is now a probability and the rows must therefore add to 1.0. This infinitesimal probability matrix $\mathbf{P}(dt)$ can be expressed in terms of the rate matrix $\mathbf{Q}$ as follows (in matrix notation):

$$\mathbf{P}dt = \mathbf{I} + \mathbf{Q}dt \tag{15.14}$$

From the definition of the derivative of a function, we have

$$\begin{aligned} \frac{d}{dt}\mathbf{P}(t) &= \frac{\mathbf{P}(t+dt) - \mathbf{P}(t)}{dt} \\ &= \frac{\mathbf{P}(t)(\mathbf{I} + \mathbf{Q}dt) - \mathbf{P}(t)}{dt} \\ &= \frac{\mathbf{P}(t)\mathbf{I} + \mathbf{P}(t)\mathbf{Q}dt - \mathbf{P}(t)}{dt} \\ &= \frac{\mathbf{P}t\mathbf{Q}dt}{dt} \\ &= \mathbf{P}t\mathbf{Q} \end{aligned} \tag{15.15}$$

Just as the derivative of e^{xt} with respect to t (for an arbitrary x) is just e^{xt} times the derivative of xt with respect to t, the derivative of $e^{\mathbf{Q}t}$ with respect to t (for the matrix $\mathbf{Q}$) is just $e^{\mathbf{Q}t}$ times the derivative of $\mathbf{Q}t$ with respect to t. Thus, if we define

(15.16)
$$\mathbf{P}(t) = e^{\mathbf{Q}t}$$

then

(15.17)
$$\frac{d}{dt}\mathbf{P}(t) = \frac{d}{dt}e^{\mathbf{Q}t} = e^{\mathbf{Q}t}\mathbf{Q} = \mathbf{P}t\mathbf{Q}$$

which accords with equation (4.15).

We now have an equation (4.16) relating our desired matrix of transition probabilities P(t) to the matrix of instantaneous rates Q, but how is eQt calculated?

It can be shown that

(15.18)
$$\mathbf{P}(t) = e^{\mathbf{Q}t} = \mathbf{U}\mathbf{D}\mathbf{U}^{-1}$$

where **U** is the matrix of eigenvectors of **Q**, and **D** is the matrix

(15.19)
$$\mathbf{D} = \begin{bmatrix} e^{1^t} & 0 & 0 & 0 \\ 0 & e^{\lambda_2 t} & 0 & 0 \\ 0 & 0 & e^{\lambda_3 t} & 0 \\ 0 & 0 & 0 & e^{\lambda_4 t} \end{bmatrix}$$

where λ_1, λ_2, λ_3 and λ_4 are the four eigenvalues of **Q**. If you are unfamiliar with matrix algebra, and specifically with the concept of eigenvalues and eigenvectors, please consult an introductory linear algebra text: most provide good discussions of these concepts (Bulmer, 1994). The specific application leading to equation (4.18) is often found under the topic of *diagonalization* (or *spectral decomposition*) of a matrix.

How is equation (4.18) useful? For all of the models used by PAUP*, it is relatively easy to obtain the eigenvalues and eigenvectors of the instantaneous rate matrix **Q**, especially if a symbolic algebra computer package such as Mathematica™ is available. Once one has expressions for the eigenvalues and eigenvectors in terms of the base frequencies, the rate parameters, and time, we must perform the matrix multiplication in (4.18) to obtain the matrix of transition equations, $\mathbf{P}(t)$. As an example, for the instantaneous rate matrix for the Jukes-Cantor model given in equation (4.11), the eigenvalues are $\lambda_1 = 0$ and $\lambda_2 = \lambda_3 = \lambda_4 = -\mu t$, and the matrix of eigenvectors is

$$\mathbf{U} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 1 & 0 & 0 & -1 \end{bmatrix}$$

The inverse of the matrix **U** is

$$\mathbf{U}^{-1} = \begin{bmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & -\frac{3}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & -\frac{3}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & -\frac{3}{4} \end{bmatrix}$$

The transition equations for the Jukes & Cantor model can thus be calculated

$$\mathbf{P}(t) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 1 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} e^{\lambda_1 t} & 0 & 0 & 0 \\ 0 & e^{\lambda_2 t} & 0 & 0 \\ 0 & 0 & e^{\lambda_3 t} & 0 \\ 0 & 0 & 0 & e^{\lambda_4 t} \end{bmatrix} \begin{bmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & -\frac{3}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & -\frac{3}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & -\frac{3}{4} \end{bmatrix}$$

which equals, after multiplying together the three matrices,

$$\mathbf{P}(t) = \begin{bmatrix} \frac{1+3e^{-\mu t}}{4} & \frac{1-e^{-\mu t}}{4} & \frac{1-e^{-\mu t}}{4} & \frac{1-e^{-\mu t}}{4} \\ \frac{1-e^{-\mu t}}{4} & \frac{1+3e^{-\mu t}}{4} & \frac{1-e^{-\mu t}}{4} & \frac{1-e^{-\mu t}}{4} \\ \frac{1-e^{-\mu t}}{4} & \frac{1-e^{-\mu t}}{4} & \frac{1+3e^{-\mu t}}{4} & \frac{1-e^{-\mu t}}{4} \\ \frac{1-e^{-\mu t}}{4} & \frac{1-e^{-\mu t}}{4} & \frac{1-e^{-\mu t}}{4} & \frac{1+3e^{-\mu t}}{4} \end{bmatrix}$$

The probability of no change in a base over time t, for the Jukes & Cantor model, is thus given by the diagonal elements, whereas all twelve remaining elements represent the probability of a change from a base to a non-identical base.

15.5.2 *Characteristics of the models available in PAUP**

The models described above are all **Markov models**. If only the current state of a stochastic process determines the probabilities of the possible states in the future, then the process is Markovian (a Markov process). For substitution models, this means that if a site has an A at time t, then the base present at that site at time $t+1$ depends only on the fact that an A was there at time t. Knowing that a C was there at time $t-1$ makes no difference. This assumption was used in equation (4.15) when we substituted $\mathbf{P}(t)\mathbf{P}(dt)$ in for $\mathbf{P}(t+dt)$: to do this we had to assume that what happened during the time period t was independent of what happened during the time period dt.

These models, because of the symmetry in their instantaneous rate matrices, are called ***time-reversible models***, since the probability of going from base i to base j is identical to the probability of going from base j to base i (i.e., the reverse direction). This can be easily shown for any one of the transitions. For example, using the general 6-parameter model, the probability of starting with an A and ending up at C after an infinitesimal time period dt is:

$$\pi_A \mathbf{P}_{AC}(dt) \quad = \quad \pi_A(\mu a \pi_C dt)$$

which is identical to the probability of beginning with a C and ending at A after time dt:

$$\pi_C \mathbf{P}_{CA}(dt) \quad = \quad \pi_C(\mu a \pi_A dt)$$

Time reversibility allows the application of Felsenstein's (1981) pulley principle, which states that the likelihood is the same regardless of the position of the root. Being able to work with unrooted trees reduces the number of distinct topologies for a given number of taxa, thus reducing the time needed for an analysis.

Another feature of these models is the prediction that base frequencies remain constant over time. If π_A, π_C, π_G and π_T are the initial frequencies, then the frequency of A after an instantaneous time period dt is:

$$\pi_A \mathbf{P}_{11}(dt) + \pi_C \mathbf{P}_{21}(dt) + \pi_G \mathbf{P}_{31}(dt) + \pi_T \mathbf{P}_{41}(dt) \tag{15.20}$$

The $\mathbf{P}_{ij}(dt)$ terms are from the general 6-parameter model are:

$$\begin{aligned}
\mathbf{P}_{11}(dt) &= 1 - \pi_C a\mu dt - \pi_G b\mu dt - \pi_T c\mu dt \\
\mathbf{P}_{21}(dt) &= \pi_A a\mu dt \\
\mathbf{P}_{31}(dt) &= \pi_A b\mu dt \\
\mathbf{P}_{41}(dt) &= \pi_A c\mu dt
\end{aligned}$$

and, substituting these into (4.20) and simplifying reveals that this expression equals πA, the initial frequency of A. Thus, the frequency of A has not changed over time. Models having this property are called ***equilibrium models***, because the frequencies of the states (bases) do not change: they are equilibrium base frequencies.

15.6 Implementation Details

This section is written for those who want to know how PAUP* implements the method of maximum likelihood. Some of the subsections assume a higher familiarity with maximum likelihood methods than the rest of the chapter, but this is the place to go if you want to know more about how likelihoods are calculated and branch lengths adjusted.

15.6.1 *Newton-Raphson iteration*

It is not possible to obtain a simple expression for maximum likelihood estimates of most of the parameters that we are interested in (branch lengths, transition:transversion ratio, etc.), and thus we must resort to iterative methods for arriving at the ML estimate. All iterative methods begin with an initial guess at the parameter's ML value. Successive iterations then result in values progressively closer to the ML value until a value sufficiently close to the ML value (or, more typically, sufficiently close to the previous value) is obtained. Iterative methods differ in the amount and type of information they use in obtaining the "next guess". A very effective tool in this regard is Newton-Raphson iteration, also known as Newton's method. The Newton-Raphson method is more efficient than many other methods because it uses the slope and curvature of the likelihood function itself to make an educated guess at the next value.

To see how the Newton-Raphson method works in general, imagine throwing a ball straight up in the air at an initial velocity v_0 of 160 feet/second. We'll round off the acceleration due to gravity (g) to -32 feet/second/second. The distance above the ground (i.e., the vertical displacement) after time t is given by:

$$f(t) = v_0 t + \frac{1}{2} g t^2 \tag{15.21}$$

and the velocity and acceleration are given by the first and second derivatives of $f(t)$

$$\begin{aligned} f'(t) &= v_0 + gt \\ f''(t) &= g \end{aligned} \tag{15.22}$$

The plot below shows the relationship between time (in seconds) and the distance from the ground graphically:

Suppose we were interested in knowing the time at which the ball stopped moving upward. This happens when the velocity of the ball is exactly zero, which is when the first derivative of $f(t)$ is zero. Even though the answer can be worked out analytically for this example, it better serves our present purpose to obtain the answer using Newton' method. Let's say our initial guess is $t = 2$ seconds, corresponding to the point labeled A in the above figure. How can we use information about the curve at point A to help us find the value for t at which the velocity of the ball equals zero? Assuming the acceleration is constant, we can predict when the ball's velocity will equal zero using the following

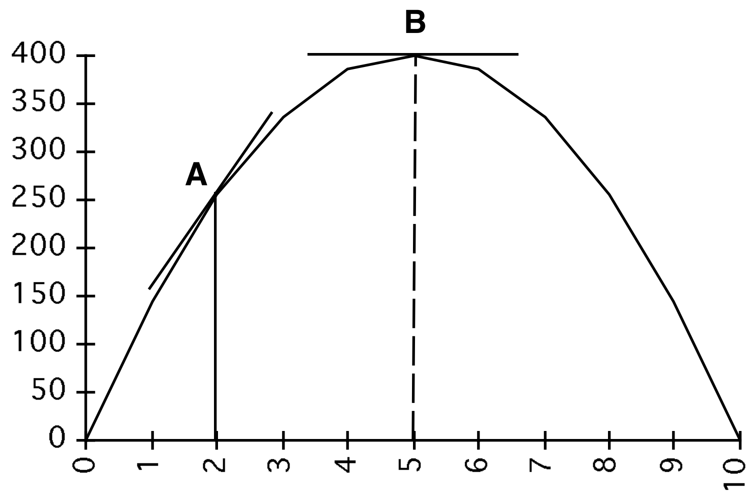


Figure 15.8 *Plot showing the distance travelled by a ball thrown straight up at 160 feet/sec as a function of time. At the point A, the slope of the curve is 96, meaning the ball is still traveling at a velocity of 96 feet/sec. At point B, the slope (=velocity) is 0, indicating that the ball is moving neither up nor down.*

simple formula:

$$t^{(i+1)} = t^{(i)} + \frac{f'(t^{(i)})}{-f''(t^{(i)})} \tag{15.23}$$

where i is the number of the current guess and $i + 1$ provides the guess for the next round of iteration (should that be necessary). Using our knowledge of the initial velocity (v^0) and the acceleration due to gravity (g), the velocity of the ball at time $t = 2$ seconds is calculated to be 96 feet/sec and our updated guess becomes:

$$\begin{aligned} t^{(1)} &= t^{(0)} + \frac{f'(t^{(0)})}{-f''(t^{(0)})} \\ &= 2.0 + \frac{96}{-32} = 5.0 \end{aligned}$$

In this admittedly contrived example, we found the peak after only one iteration, but in cases where the acceleration is not constant, it can takes several rounds of iteration before convergence occurs. The first derivative is termed the ***score***, and the negative of the second derivative the ***information*** (Edwards, 1972; Weir, 1990). Thus, for any continuously differentiable function (i.e., there are no places where the function shoots off into infinity), we can always get an updated estimate of a parameter by adding to it the score divided by the information at that point.

We can use the Newton-Raphson method for estimating parameters in likelihood models by simply letting the likelihood curve take the place of the distance curve in the figure above, and letting the parameter of interest form the x-axis instead of time. The score is just the first derivative of the likelihood *with respect to the parameter of interest*. Likewise, the information is the negative of the second derivative of the likelihood with respect to the parameter of interest. See the sections below on estimating branch lengths for more information on the use of Newton-Raphson iteration in estimating model parameters.

15.6.2 *Estimation of base frequencies*

Base frequencies in PAUP* are estimated as follows:

(15.24)
$$\pi_i = \frac{1}{nm}\sum_{i=1}^{n}\sum_{i=1}^{m} x_{ij}$$

where *n* is the number of taxa and *m* the number of sites and

$$x_{ij} = \begin{cases} 1 & \text{if the base at site j in taxon i is k} \\ 0 & \text{otherwise} \end{cases}$$

The total, *nm,* does not include missing data points. While these frequencies are not maximum likelihood estimates, they are close enough to the MLEs (Goldman, 1993) that it is generally not worth the considerable extra computational effort to iterate the frequencies along with the branch lengths and other model parameters.

A problem arises when there are ambiguites in the data. If, for example, it is not clear for a particular site whether the base is an *A* or a *G*, we must add some amount to the current *A* count and some amount to the current *G* count, and these amounts must add to one because, even though there is ambiguity, there really can be only one base at that site. If we knew all the bases were equally frequent, we could simply add 0.5 to the count of *A* nucleotides and 0.5 to the count for *G* nucleotides; however, base frequencies are seldom exactly equal and the best strategy is to weight the amount added to each appropriate count by the current best estimate of its frequency. Thus, if the frequency distribution was, in reality,

$$\pi_A = 0.1 \quad \pi_G = 0.2 \quad \pi_C = 0.3 \quad \pi_T = 0.4$$

we should add 0.1 / 0.3 = 1/3 to the count of *A* nucleotides and 0.2 / 0.3 = 2/3 to the count of *G* nucleotides. Since we don't know what the true nucleotide frequencies are (after all, estimating them is the purpose of this little exercise), we must use our current best estimate of the frequency distribution and iterate until we get convergence. PAUP* iterates until the sum of the deviations for all four base frequencies is less than $1x10^{-8}$.

15.6.3 Computing likelihoods of trees

For time reversible models (i.e., all those implemented in PAUP*), because of the "pulley principle" the position of the root can be anywhere on the tree (Felsenstein, 1981a). For the purpose of computing likelihoods, then, we can place the root at one end of a given branch (call it *b*), calling the end at the root the *proximal* end of the branch, and the other end the *distal* end. The likelihood of the tree for site *k*, in the absence of site-to-site rate heterogeneity, can then be written

$$L_k = \sum_i \pi_j L_j^{(prox)} \sum_j L_j^{(dist)} P_{ij} \nu \tag{15.25}$$

where *i* is the base at the proximal end, *j* is the base at the distal end, and pi is the frequency of the base at the root. $L_i^{(prox)}$ and $L_j^{(dist)}$ are termed conditional likelihoods because they represent the likelihood from some point (either the proximal or distal node) away from the root, *conditional* *on* the rest of the tree. $L_i^{(prox)}$ is that portion of the likelihood resulting from consideration of all branches on the proximal side of branch b conditional on the base present at the proximal node. $L_j^{(dist)}$ is that portion of the likelihood resulting from consideration of all branches on the distal side of branch *b* conditional on the base at the node at the distal end of branch *b*. This can be illustrated from consideration of the five-taxon tree below:

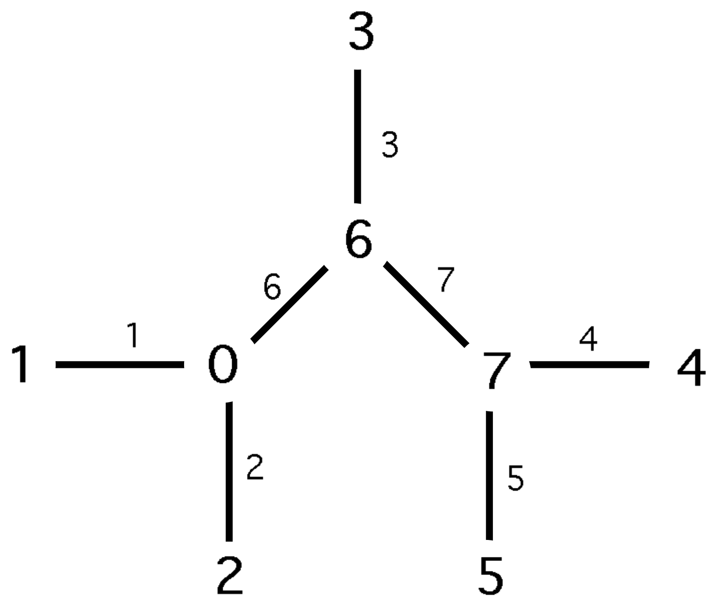


Figure 15.9 *5 taxon unrooted tree.*

If *b* is branch 6 and node 0 is the root, then $L_i^{(prox)}$ is that part of likelihood pertaining to branches 1 and 2 and conditional on the state (i) at the root node 0:

$$L_i^{(prox)} = P_{i,s_1}(v_1)P_{i,s_2}(v_2) \tag{15.26}$$

and $L_j^{(dist)}$ is that part of the likelihood pertaining to branches 3, 4, 5, and 7, conditional on the state (j) at the distal node (node 6) of branch 6:

$$(15.27) \qquad L_j^{(dist)} = P_{j,s_3}(v_3)P_{j,s_7}(v_7)\sum_{s_7} P_{s_7,s_4}(v_4)P_{s_7,s_5}(v_5)$$

where s_u is the state at node u, and the $P_{i,j}(v)$ terms represent transition probabilities from state i to state j across a branch of length v. Since computing these conditional likelihoods constitutes the bulk of the computational effort, PAUP* keeps, for each node of the tree, two arrays of conditional likelihoods in memory (one for proximal and one for distal conditional likelihoods) so that only those that have been made invalid (by a change in a branch length, for example) need be computed. When a branch's length changes during estimation of its length, all conditional likelihoods depending on that branch must be recomputed.

Under site-to-site rate heterogeneity, there are m rate categories to consider for each site when computing the likelihood:

$$(15.28) \qquad L_k = \sum_{r=1}^{m} Pr(r)L_k^{(r)} = \sum_{r=1}^{m} \frac{1}{m} \sum_i \pi_i L_i^{(prox,r)} \sum_j L_j^{(dist,r)} P_{ij}^{(r)}(v)$$

where $1/m$ is the probability that site k has relative rate r (rates are equiprobable) and the transition probabilities are now conditional on rate r (making the conditional likelihoods also conditional on rate r). Our proximal and distal arrays of conditional likelihoods at each node of the tree must now be m times larger than before to accommodate conditional likelihoods for all m relative rates. Note that if all m relative rates are 1.0, then the above formula collapses, as it should, to the likelihood under the assumption of homogeneity of rates. Since the likelihood for any given site now involves a sum over m rates, the entire analysis can be expected to take at least m times longer to complete. A more complete description of likelihoods under a discrete gamma model of rate heterogeneity can be found in Yang (1994b).

15.6.4 *Estimation of branch lengths*

Ideally, all of the parameters in a model would be simultaneously updated using a multidimensional Newton-Raphson method (see Weir, 1990, pp. 281-283). In the interest of speed, PAUP* updates each branch length parameter separately, using the one-dimensional Newton-Raphson method described above. This could potentially cause two problems: 1) we could iterate into a local, rather than global, optimum branch length combination, or 2) iteration is inefficient and resulting branch length estimates are incorrect because of premature termination of the iterations. The first problem does not appear to occur very often with real data (Muse and Gaut, 1994), especially if the initial branch length estimates are good. The second problem results from the fact that the curvature of the likelihood surface for one particular branch length can vary depending on the values of the other parameters. If one of the other branch lengths is currently set to a value that is some distance away from its MLE, then we cannot expect to get good update information for the branch length of interest. This can lead to a lot of overshooting and undershooting, which means that we approach the MLE much more slowly. No widely available program for inferring maximum likelihood phylogenies uses the multidimensional Newton-Raphson approach because of the necessity of inverting the information matrix, which has at least $2N - 3$ rows and columns, where N is the number of taxa.

PAUP* begins by estimating the branch lengths using either Fitch-Margoliash method with Jukes-Cantor distances or the Rogers-Swofford method (described in the section "Rogers-Swofford Branch Length Estimation"). In order to update one particular branch length (say, branch *b*), we consider the tree to be rooted at one end of branch *b* (the proximal end, the distal end being the end furthest away from the root) as in the discussion of computing likelihoods (above). What we need for Newton-Raphson iteration are the first and second derivatives of the ln-likelihood (the natural logarithm of the likelihood function) with respect to branch *b*'s length. The branch length parameter is the time, *t*; however, the unit in which time is measured is not years or generations, but expected numbers of substitutions. We arrive at such a scaling for t by setting the rate parameters in such a way that when$t = 1$, the product of the overal rate of substitution and time is also 1. For example, for the Kimura (1980) 2-parameter model, the expected number of substitutions ("Transition equations") is

$$v = \frac{\kappa + 2}{4}\beta t \tag{15.29}$$

so, since *k* is fixed, setting both *t* and *v* to one allows us to compute

$\beta = 4/(\kappa + 1)$. By making b a constant, we have effectively combined the overall substitution rate and time parameters into just the time parameter. Since the ln-likelihood is

$$\ln L = \sum_k c_k \ln L_k \tag{15.30}$$

whereL_k is the likelihood for the kth pattern and c_k is the number of sites showing pattern k, the first derivative of the ln-likelihood with respect to t (the score) is just

$$\frac{\partial}{\partial t} \ln L = \sum_k \left(\frac{c_k \frac{\partial}{\partial t} L_k}{L_k} \right) \tag{15.31}$$

where

$$\frac{\partial}{\partial t} L_k = \sum_i \pi_i L_i^{(prox)} \sum_j L_j^{(dist)} \frac{\partial}{\partial t} P_{ij}(t) \tag{15.32}$$

The terms $L_i^{(prox)}$ and $L_j^{(dist)}$ are the conditional likelihoods for pattern k that are stored in two arrays associated with branch b. These conditional likelihoods are kept up to date in that, whenever any branch length changes, all affected conditional likelihoods are immediately recomputed. Likewise, the second derivative of the ln-likelihood with respect to t (the negative of the information) is

$$\frac{\partial^2}{\partial t^2} \ln L = c_k \sum_k \left[\frac{\frac{\partial^2}{\partial t^2} L_k}{L_k} - \left(\frac{\frac{\partial}{\partial t} L_k}{L_k} \right)^2 \right] \tag{15.33}$$

where

$$\frac{\partial^2}{\partial t^2} L_k = \sum_i \pi_i^{(prox)} \sum_j L_j^{(dist)} \frac{\partial^2}{\partial t^2} P_{ij}(t) \tag{15.34}$$

Thus, to compute the score and information we need for doing Newton-Raphson iteration we can get away with using the same machinery used to compute the ln-likelihood, simply using derivatives of $P_{ij}(t)$ instead of $P_{ij}(t)$ itself. A crude algorithm[*] is presented below in C, where `pi` is an array holding the base frequencies and `b` is a structure holding the data (length and conditional likelihood arrays) for branch *b*. The functions `P`, `dP`, and `d2P` return the transition probability and first and second derivatives of the transition probability, respectively.

[*]This algorithm is much slower than the routines actually employed in PAUP*; however, this version is presented here because it is much easier to follow and the end results are the same

```
double L, dL, d2L, Lk, dLk, d2Lk, x;
int k, i, j;
L = dL = d2L = 0.0;
for( k = 0; k < num_patterns; k++)
{
     x = Lk = dLk = d2Lk = 0.0;
     for( i = 0; i < 4; i++)
     {
          for( j = 0; j < 4; j++)
          {
               x = pi[i] * b-> Lprox[k][i] * b-> Ldist[k][j];
               Lk += x * P(i, j, b-> );
               dLk += x * dP(i, j, b-> );
               d2Lk += x * d2P(i, j, b->);
          }
     }
     L += c[k] * log( Lk );
     dL += c[k] * ( dLk / Lk );
     d2L += c[k] * ( d2Lk / Lk - (dLk / Lk) * (dLk / Lk) );
}
```

The transition equations are presented for all models in "Transition equations."

The extension of the above to the case of rate heterogeneity is not difficult. The algorithm is the same save for the fact that the likelihood for pattern *k* now involves a loop over the *m* rate categories:

$$\begin{aligned}
L_k &= \frac{1}{m}\sum_r L_k^{(r)}, \\
\frac{\partial}{\partial t}L_k &= \frac{1}{m}\sum_r \frac{\partial}{\partial t}L_k^{(r)}, \\
\frac{\partial^2}{\partial t^2}L_k &= \frac{1}{m}\sum_r \frac{\partial^2}{\partial t^2}L_k^{(r)}
\end{aligned} \tag{15.35}$$

The modified example code for obtaining the ln-likelihood and derivatives in the case of rate heterogeneity looks like this:

```
double L, dL, d2L, Lk, dLk, d2Lk, x;
int k, i, j;
L = dL = d2L = 0.0;
```

```
for( k = 0; k < num_patterns; k++ )
{
for( r = 0; r < num_rates; r++ )
     {
          x = Lk = dLk = d2Lk = 0.0;
          for( i = 0; i < 4; i++)
          {
               for( j = 0; j < 4; j++ )
               {
                    x = pi[i] * b-> Lprox[r][k][i]
                                 * b->[r][k][j];
                    Lk += x * P(i, j, b->, r);
                    dLk += x * dP(i, j, b->, r);
                    d2Lk += x * d2P(i, j, b->, r);
               }
          }
          L += c[k] * log( Lk );
          dL += c[k] * ( dLk / Lk );
          d2L += c[k] * ( d2Lk / Lk - (dLk / Lk) * (dLk / Lk) );
     }
}
```

The formulas presented in "Transition equations" for the transition equations all include the parameter relative rate, r. When $r = 1$, these formulas all reduce to their rate-homogeneity counterparts.

For the general 6-parameter model, we can represent the matrix (**P**) of transition probabilities in terms of a matrix of eigenvectors (**U**) of the instantaneous rate matrix and a matrix (**D**) having diagonal elements in the form $e^{\lambda t}$, where the λ are eigenvalues of the instantaneous rate matrix:

$$\mathbf{P} = \mathbf{U}\mathbf{D}\mathbf{U}^{-1} \qquad (15.36)$$

Doing the matrix multiplication for a specific transition probability $P_{ij}(t)$, we find that

$$p_{ij}(t) \quad = \quad \sum_{k=0}^{4} u_{ik} v_{kj} e^{-\lambda_k rt} \qquad (15.37)$$

where u_{ik} is the kth element from the ith row of $\mathbf{U}$, and v_{kj} is the kth element of the jth column of $\mathbf{U}^{-1}$. It is easy to find the first and second derivatives of $P_{ij}(t)$ with this formulation

$$\begin{aligned}\frac{\partial}{\partial t} &= -\sum_k u_{ik} v_{kj} \lambda_k r e^{-\lambda_k r t} \\ \frac{\partial^2}{\partial t^2} &= \sum_k u_{ik} v_{kj} \lambda_k^2 r^2 e^{-\lambda_k r t}\end{aligned} \tag{15.38}$$

(where $r = 1$ for rate homogeneity) and these quantities can be used, as above, to obtain the score and information needed to update a particular branch length.

Updating the branch length is easy once we have the score and information in hand:

$$\begin{aligned}t' &= t + \frac{score}{information} \\ &= t - \frac{\frac{\partial}{\partial t} \ln L}{\frac{\partial^2}{\partial t^2} \ln L}\end{aligned} \tag{15.39}$$

Using the sample code again, we need only add the following line after the end of the loop over patterns:

```
b-> -= dL / d2L;
```

15.6.5 Estimation of other model parameters

Although the Newton-Raphson method provides an efficient method for optimizing branch lengths, the method is not as useful for estimating other parameters of the models of DNA substitution assumed in PAUP* because the first and second derivatives with respect to these other parameters cannot easily be computed. Hence, PAUP* uses "derivative free" methods to find maximum likelihood estimates of the gamma shape parameter, the proportion of invariable sites, kappa (the transition / transversion rate ratio), the relative substitution rates of different data subsets, and/or the rate parameters of the 6-parameter model of DNA substitution. Furthermore, when a molecular clock is assumed, a derivative free method is used to estimate the branch points on the tree. Details of the algorithms implemented in PAUP* can be found in Brent (1973). Specifically, a modification of the Powell (1964) method described in Brent (1973, chapter 7) is used when several parameters are jointly estimated and the Brent (1973, chapter 5) method is used when only a single parameter, besides the branch lengths, is estimated.

15.6.6 Rogers-Swofford Branch Length Estimation

The Rogers-Swofford method (Rogers and Swofford, 1998) is a method of estimating initial branch lengths for further optimization by Newton-Raphson iteration. The method uses the parsimony criterion to reconstruct the changes at the internal nodes of a candidate tree (assuming unordered character-state transformation). Ambiguities in character state assignments are resolved by choosing that reconstruction which forces a reconstructed change to the longer branch of the phylogeny, because doing so has a smaller effect on the likelihood of the tree than placing the change along a short branch. The method appears to provide branch lengths that are generally closer to the maximum likelihood estimates than does the Fitch and Margoliash (1967) method.

15.7 History of ML phylogeny inference

The method of maximum likelihood is one of the most generally useful methods for obtaining estimates of the parameters in a statistical model and for testing hypotheses based on such models. The use of likelihoods for estimating parameters and comparing competing hypotheses was, like much of the field of statistics, invented by R. A. Fisher (Edwards, 1972; Fisher, 1922). Statistical inference using likelihood methods is now widespread in many fields of science and is not by any means restricted to Fisher's field of primary interest (biology, specifically genetics). The first proposed application of likelihood methods to phylogenetic inference was for gene frequency data in a series of papers begun by Edwards and Cavalli-Sforza (1964), although likelihood methods were not actually implemented in those studies because of complications resulting from the complexity of their model. Neyman (1997) discussed the application of likelihood to nucleotide sequence data in 1971, however the first algorithms for the case of more than three species were proposed (still for gene frequency data) by Felsenstein (1973) and Thompson (1975), who made likelihood more practical by employing simpler models of evolution. Much of the remaining history of the use of likelihood methods in phylogenetics can be read from the publications of Joe Felsenstein, who described likelihood methods for estimating evolutionary trees from continuous characters (Felsenstein, 1973), DNA sequences (Felsenstein, 1981a), and restriction site data (Felsenstein, 1992), and for weighting characters for phylogenetic inference (Felsenstein, 1981b). Felsenstein also wrote and continues to maintain the computer package (PHYLIP) for maximum likelihood phylogenetic inference (Felsenstein, 1991). Work in the last decade has focused primarily on the development of models that incorporate more of what is known about the evolutionary process (e.g., Goldman and Yang, 1994; Hasegawa et al., 1985; Kishino et al., 1990; Thorne et al., 1991, 1992; Thorne and Kishino, 1992; Tillier and Collins, 1994; Yang, 1993, 1994b), application of likelihood ratio tests to specific hypotheses (e.g., Felsenstein, 1988; Muse and Gaut, 1994; Muse and Weir, 1992), and on ways to improve the adequacy of models for a given data set (e.g., Bishop and Friday, 1985; Goldman, 1993; Yang et al., 1994). Models for the evolution of morphological characters have also been developed for use within a maximum likelihood framework (Pagel, 1994), although such models have been used primarily for testing specific hypotheses concerning correlated evolution rather than for inferring phylogeny (but see Lewis, 2001).

15.8 Transition equations

The transition equations are provided below for all models available for ML inference in PAUP*. Also given are concise formulas for branch lengths in terms of expected numbers of substitutions, given the rate parameters, base frequencies, and time. The parameters used in the equations are those used to define the instantaneous rate matrices in the text. Also, the symbol r has been used to denote the relative rate used in modeling site-to-site rate heterogeneity. When r is set to 1, the equations reduce to versions that assume homogeneity of substitution rate across sites. To reduce the size of some of the matrices, the symbols πY and πR are used instead of $\pi C + \pi T$ and $\pi A + \pi G$, respectively. Also, the symbol Π_i means πR if base i is a purine (A or G) or πY if base i is a pyrimidine

(*C* or *T*). Other space-saving substitutions are noted in the individual sections in which they are used.

15.8.1 Jukes & Cantor, 1969 (JC)

Expected number of substitutions

$$\mu = \frac{3}{4}\mu r t \tag{15.40}$$

Transition equations

$$P_{ij} = \begin{cases} \frac{1}{4} + \frac{3}{4}e^{\mu r t} & i = j \\ \frac{1}{4} - \frac{1}{4}e^{\mu r t} & i \neq j \end{cases} \tag{15.41}$$

Settings for model in PAUP*

- `lset nst=1 basefreq=equal;`

15.8.2 Felsenstein, 1981a (F81)

Expected number of substitutions

$$\mu = (-\sum \pi_i^2 \mu r t \tag{15.42}$$

Transition equations

$$P_{ij} = \begin{cases} \pi_j + (1 - \pi_i)e^{-\mu r t} & i = j \\ \pi_j(1 - e^{-\mu r t}) & i \neq j \end{cases} \tag{15.43}$$

Settings for model in PAUP*

- `lset nst=1 basefreq=estimate;`

15.8.3 *Kimura, 1980 (K2P)*

Expected number of substitutions

(15.44) $$\mu = \frac{\kappa+2}{4}\mu t$$

Transition equations

(15.45) $$P_{ij} = \begin{cases} \frac{1}{4} + \frac{1}{4}e^{-\mu rt} + \frac{1}{2}e^{-\mu rt\left(\frac{\kappa+1}{2}\right)} & i = j \\ \frac{1}{4} + \frac{1}{4}e^{-\mu rt} - \frac{1}{2}e^{-\mu rt\left(\frac{\kappa+1}{2}\right)} & i \neq j, transition \\ \frac{1}{4} - \frac{1}{4}e^{-\mu rt} & i \neq j, transversion \end{cases}$$

Settings for model in PAUP*

- `lset nst=2 TRatio=estimate basefreq=equal;`

15.8.4 *Hasegawa, Kishino, & Yano, 1985 (HKY)*

Expected number of substitutions

(15.46) $$\mu = 2\beta t[\pi_R\pi_Y + \kappa(\pi_A\pi_G + \pi_T\pi_C)]$$

Transition equations

(15.47) $$P_{ij} = \begin{cases} \pi_j + \pi_j\left(\frac{1}{\Pi_j} - 1\right)e^{-\mu rt} + \left(\frac{\Pi_j - \pi_j}{\Pi_j}\right)e^{-\mu rtA} & i = j \\ \pi_j + \pi_j\left(\frac{1}{\Pi_j} - 1\right)e^{-\mu rt} - \left(\frac{\pi_j}{\Pi_j}\right)e^{-\mu rtA} & i \neq j, transition \\ \pi_j(1 - e^{-\mu rt}) & i \neq j, transversion \end{cases}$$

Settings for model in PAUP*

- `lset nst=2 TRatio=estimate basefreq=estimate`
  `Variant=HKY;`

15.8.5 *Felsenstein, 1984 (F84)*

Expected number of substitutions

(15.48)
$$\mu = \left[(1 - \sum_i \pi_i^2) + \kappa \left(1 - \sum_i \frac{\pi_i^2}{\Pi_i} \right) \right]$$

Transition equations

(15.49)
$$P_{ij} = \begin{cases} \pi_j + \pi_j \left(\frac{1}{\Pi_j} - 1 \right) e^{-\mu rt} + \left(\frac{\Pi_j - \pi_j}{\Pi_j} \right) e^{-\mu rt(\kappa+1)} & i = j \\ \pi_j + \pi_j \left(\frac{1}{\Pi_j} - 1 \right) e^{-\mu rt} - \left(\frac{\pi_j}{\Pi_j} \right) e^{-4\mu rt(\kappa+1)} & i \neq j, transition \\ \pi_j (1 - e^{-\mu rt}) & i \neq j, transversion \end{cases}$$

Settings for model in PAUP*

- `lset nst=2 TRatio=estimate basefreq=estimate Variant=F84;`

16 Distance Based Analyses

Peter J. Waddell, David L. Swofford, and Paul O. Lewis.

16.1 Introduction

This chapter deals with the theory and applications of using pairwise distances between sequences in order to estimate a tree. Specific information on how to use the program is given in later chapters. The transformed distance based analyses in PAUP* are model based and intended specifically for the analysis of DNA, RNA and Protein sequences (including 2 state sequences e.g. R / Y coding).

This chapter begins with a brief introduction to exploratory methods, and why distance based tree building can excel in this area. Quite detailed explorations of the patterns of evolution and its implications for phylogenetics can be completed in a short time with distance based methods. Distance methods offer analogues of many of the tests that can be made with ML.

The most useful comments on phylogenetic practice are usually made as one method compared to another. This is because we do not know what the true model of evolution is, and additionally are only beginning to understand some properties of tree estimation procedures. To make these relative comparisons sensibly, it is often necessary to understand the breadth of the methods available. This is the approach taken in this chapter where we introduce the theory and practice of distance based phylogenetic analysis in a wide sense, so that readers can best appreciate the attributes of methods currently available in PAUP*. Further, in the next few years, it is envisaged that many more of the methods and techniques described here will be implemented in PAUP*.

16.2 Distance based phylogenetic methods make for practical data exploration techniques

It is our contention that distance based phylogenetic methods are presently the best exploratory methods for use in molecular phylogenetics, and this is a role which should not be overlooked. So what are exploratory methods:

”If we need a short suggestion of what exploratory data analysis is, I would suggest that

1. It is an attitude, AND

2. A flexibility, AND

3. Some graph paper (or transparencies or both).

”No catalogue of techniques can convey a willingness to look for what can be seen, whether or not anticipated.” (Tukey, 1980, p. 24).

Distance analyses are arguably the best data exploratory phylogenetic techniques available at the moment. This is because they are most varied in their assumptions, quick to apply (especially with the important technique of bootstrapping), capable of dealing with large data sets, and often sensitive enough to show up systematic biases before other methods. In addition, distance based methods have been developed which allow a useful graphical presentation of major trends in the data [(e.g. the distance Hadamard (Hendy and Penny, 1993), or split decomposition, (Bandelt and Dress, 1992)], a factor often missed when studying reconstructed trees. Thus, distance based methods have a unique position which is presently not seriously challenged by ML.

Data exploratory techniques, as Tukey (1980) discusses, are preferably methods which allow the researcher to effectively unveil information in the data. To do this they must be flexible, fast enough to be interactive (in the sense that they allow many quick exploratory analyses), and informative. Distance based methods presently do this admirably. Through their ability to analyse data transformed under the widest variety of models, they can inform us of the stability of the estimated tree in the light of obvious features of the sequence data. Studying how distance based tree estimates and associated bootstrap support change as we attempt to correct for different evolutionary factors are crucial steps in any analysis.

The distance transformations considered in this chapter make model based corrections for known features of sequence evolution. While ML also do this, distance based methods express a mixture of both strengths and weaknesses relative to ML. Strengths include being much faster computationally. This not only allows much larger data sets to be analysed, but perhaps more importantly more intensive analysis of moderate sized data sets. This includes looking at the differences made by a wide range of testable assumptions about the evolutionary process (e.g. Ti / tv ratio, unequal base compositions, non-stationary base compositions, unequal rates across sites, using just transversions etc.). These factors are inferred from the results of the analyses after transforming the distances to take into account additional assumptions. In each one of these analyses it is usually feasible to make not just a point estimate (the best tree) but also an estimate of the sampling error (e.g. via the bootstrap). This assists in identifying which alterations in a tree deserve special mention, and which are perhaps best dealt with by pointing out stochastic variability and probably require more data.

Drawbacks of distance methods include the theoretical expectation that with longer sequences ML methods will make more statistically efficient use of the data, and so give rise to smaller stochastic errors (Kuhner and Felsenstein, 1994). Additionally, there is evidence that ML methods may be more robust than distance methods with regard to many but not necessarily all deviations from the model (Fukami and Tateno, 1989; Hasegawa and Fujiwara, 1993; Waddell, 1995). However, even these apparent drawbacks can be turned to an advantage. The increased sensitivity to systematic errors can highlight features of the evolutionary process that may go unnoticed in an ML analysis, so a contrast between these methods can be very enlightening.

There are a wide variety of tree selection criteria and algorithms available for analysing distance data. While practically all of them are guaranteed to work with additive data, they will differ in their computational speed, robustness and statistical efficiency. When they may differ in the trees they select as optimal with real data, can also serve as a additional warning that the optimal tree should not be considered to the exclusion of all others. Indeed one method, split decomposition, has the potential of showing where in a tree systematic bias may be clearly and strongly acting. Although not currently implemented in PAUP*, there is a splits trees program which will take input of distance matrices directly from PAUP* via nexus files.

Lastly, very general (and hopefully robust) distance estimates are now available (particularly the LogDet combined with constant site removal) which presently have no counterpart in the available ML programs. These distances tend to show some surprisingly good sampling properties, can make appreciable differences to analyses made now, and would be much faster than their ML counterparts if available.

16.3 Estimating trees given a matrix of pairwise distances

Mathematically a tree is a graph with no circuits in it (that is there is only ever one route to get from point a to point b). While dry, this definition immediately has a strong consequence ”The additive pairwise distances measured on a weighted tree perfectly define that tree and only that tree.” By a weighted tree we mean a tree with all the edge lengths given, while additivity is a property of a distance measure on a graph. Additivity implies that if we put an arbitrary node (x) anywhere in our weighted tree on the path from tip a to tip b, then the defined distance from one tip (a) to (b), or $d_{ab} = d_{ax} + d_{xb}$.

A commonly used additive distance measure is the total number of substitutions along each edge, or along each path in the tree. This measure is often rescaled to the average number of substitutions per site. An interesting property of distance measures, is that if you go through our tree and weight each substitution differently, a distance measure counting all these weighted substitutions will still be additive only on the original tree, however, edge lengths will now be in a weighted number of substitutions per site. This factor becomes important later in understanding how to best interpret the very general LogDet distances (see below).

With perfectly additive distances, it is quite trivial to use them to reconstruct the weighted tree. There have been many algorithms developed to build a tree from additive distances (Felsenstein, 1988; Swofford et al., 1996). A few of the more well known are the distance Wagner algorithm (Farris, 1972), ST neighborliness algorithm (Sattath and Tversky, 1977), and neighbor-joining algorithm (Saitou and Nei, 1987). All work on simple algebraic principals. None of these algorithms assume equal evolutionary rates or clock-like behaviour, and almost all are consistent given additive distances (of course, given non-additive distances they will may show different weaknesses). It is also possible to devise tree building algorithms that assume that there is a perfect clock (i.e. ultrametric distances) and such a method is the unweighted pair group using mathematical averages (UPGMA). This is a simple clustering method that can be found in other statistical applications (Sokal and Rohlf, 1981). However, for various reasons, it does not appear to perform well in phylogenetics (see the following section on UPGMA).

To reinforce the concert of additivity and show some of its emergent properties, see figure 1. Notice how the additive distances fit perfectly onto one weighted tree. The edge lengths could be the expected number of substitutions per site, or a weighted number of substitutions per site (as will be described later for the LogDet distances).

Figure 1. Additivity of distances on a tree.

A simple consequence of additivity on a tree of four taxa is the Buneman (1971) 4-point metric:

$d_{AB} + d_{CD} < d_{AC} + d_{BD}$, and $d_{AC} + d_{BD} = d_{AD} + d_{BC}$

since (substituting in the edge lengths):

$(e_1+e_2)+(e_3+e_4) < (e_1+e_5+e_3)+(e_2+e_5+e_4) = (e_1+e_5+e_4)+(e_2+e_5+e_3),$

then canceling terms gives:

0 < 2e5 = 2e5,

i.e. the other pairs of distances count the internal edge length twice.

This simple metric will be used next to help explain what can happen when distances become non-additive. An interesting property of many tree estimation methods based on distances is that with just four taxa, they will always chose the tree identified by the minimum sum of $d_{ij} + d_{kl}$. These methods include neighbor-joining, a popular kind of minimum evolution (ME (OLS)), and the ST method (these methods are described further below). The ST method, is an algorithmic method which attempts to maximise the number of sets of four taxa in the whole tree that meet the expectation that $d_{ij} + d_{kl}$ is minimal [Fitch (1981) later called this criteria maximising neighborliness]. While other distance methods (e.g. weighted least squares) do not follow this expectation exactly, with four taxa they may not deviate from it much. Overall then, this property is an important one to grasp for the further understanding of tree estimation. When observed or transformed distances are expected to violate Buneman's four point metric due to some systematic bias, tree inference based on these distances faces the real danger of becoming inconsistent (or 'positively misleading', i.e. the more data is accumulated, the more the optimal tree is likely to be other than the 'true' tree).

16.4 Two potential causes of inconsistency with distance based tree estimation

Before we go into tree estimation and distance correction in more detail, it is important to understand that distance based methods can suffer not only the 'long edges attract' form of inconsistency, but also a potential problem of 'long edges repel' if overestimation of larger distances occurs. A similar problem can also occur with ML, and parsimony applied to transformed data (Waddell, 1995). Thus, distance based methods can potentially go wrong in either direction, while standard unweighted parsimony can goes wrong in just one direction (long edges attract). This is itself not a sign of weakness but rather of flexibility, yet it does require attention and consideration with some of the wide variety of distance transforms now available.

16.5 Long Edges attract: a problem for distances also

The original 'long edges attract' problem of inconsistency (getting the wrong answer with arbitrarily long sequences) was described by Felsenstein (1978a), with the term 'long edges attract' coined by Hendy and Penny (1989) when they showed the same problem could occur under a molecular clock. In the case of parsimony and four taxa, it is best understood in terms of the probability of observing site patterns where the two long edges end up having the same state by multiple substitutions. This probability can exceed that of the pattern generated directly by the internal edge (if it is short enough, e.g. see Swofford et al. (1996) for a worked example). If the long edges should not be grouped together, then this factor can mislead parsimony, so that it becomes inconsistent (i.e. with infinite data it would be guaranteed to get the wrong answer).

Figure 2. The Felsenstein and anti-Felsenstein trees. Note, there is a second possible Felsenstein tree preserving the assignment of long and short edges, that is interchanging taxa 1 and 2.

In the case of distances, an alternative description of inconsistency is useful. As we have seen earlier, the four-point metric is an expected property of distances on a tree, and it leads to a simple tree selection criterion, "pick the tree, T_{ij}, which minimises $d_{ij} + d_{kl}$." Progressive underestimation of the true distance (e.g. due to multiple hits, see figure 3), reduces the gap between ($d_{13} + d_{24}$), which should be minimal on the Felsenstein tree shown, and the support for the other two trees ($d_{12} + d_{34}$) and ($d_{14} + d_{23}$). The sum which is most rapidly decreasing is $d_{12} + d_{34}$, as this contains the largest distance which proportionally is the most underestimated. If this underestimation is large enough, then the sum $d_{12} + d_{34}$ may become minimal if the internal edge is not large. Then we will pick the wrong tree (in this case $d_{12} + d_{34}$ would be minimal, so taxa 1 and 2 will be grouped together as with parsimony).

To quantify this effect: inconsistency of tree selection via the four point-metric will occur when "the total underestimation of additive distances for $d_{12} + d_{34}$ becomes greater than the total underestimation of distances $d_{13} + d_{24}$ plus twice the length of the internal edge" (or the same statement, except replacing $d_{13} + d_{24}$ with $d_{14} + d_{23}$ to represent the third tree). Considering figure 3, it is easy to see that the differential underestimation of distances can quite quickly become significant (e.g.

using observed distances when an additive distance follows the curve for a gamma (Γ) distribution of rates across sites).

The long edges attract effect can only result in inconsistency on a Felsenstein type of tree (i.e. a tree where the maximum distance is between species which are not grouped together). Notice, however, that if we consider all the 4-taxon trees with random edge lengths, there are twice as many Felsenstein trees as anti-Felsenstein trees. In contrast, the long edges attract problem will be felt on the anti-Felsenstein tree as increasingly high bootstrap support, which may not accurately reflect the statistical support for this tree under a better model. Conversely, even before inconsistency occurs for a Felsenstein tree, the bootstrap support is expected to drop appreciably due to long edges attract, and may result in a well supported phylogenetic hypothesis being passed over as unresolved.

16.5.1 Long edges repel and the anti-Felsenstein zone

Consider now what would happen with differential overestimation of distances (so that the proportion of overestimation increased with the true distance). To put it in more concrete terms: What could happen if the true distance followed a Markov model with identical site rates (e.g. the i.r. curve in figure 3) but you decided to use a G distance with shape parameter $a = 1$, because someone had claimed it to be a good default value to use? Well on a Felsenstein tree, it would give an inflated estimate of the support for that tree, as it pushed the long edges apart (Waddell, 1995), by making the sum of $d_{12} + d_{34}$ and $d_{14} + d_{23}$ on figure 2a larger than they should be.

However, on the anti-Felsenstein tree, progressive overestimation of distances can result in inconsistency (for our purpose an anti-Felsenstein tree can be any 4-taxon tree where the largest distance is between two taxa grouped on the tree). An easy way of understanding this is to first recognise that the largest distance can rapidly be transformed by a curves like those in figure 3, so it becomes arbitrarily bigger than the second largest distance (e.g. near the asymptote of an invariant sites curve). Thus, it is guaranteed that you can make the sum of $d_{12} + d_{34}$ non-minimal, irrespective of the size of the internal edge, because d_{34} is the largest distance on the anti-Felsenstein tree shown (fig. 2). This factor is called 'long edges repel' (Waddell, 1995), and often occurs when there is ’overcorrection’ of the data. It can affect all model based methods including ML (Waddell, 1995). Clearly then, this to is a factor we should bear in mind when considering the properties of transformed distances, i.e. be cautious that distance transformations are not inferring some distances which are proportionally too large.

16.6 Dealing with distances estimated from finite data and real evolutionary processes

The theory of tree estimation based on perfectly additive sequences is fine in principal, but now it must be applied to real sequences. Here the complications begin. Firstly, all distances measured on finite sequences are subject to sampling error. This source of error tends to zero as sequences become long. However, for the sequence lengths commonly used in phylogenetic analyses (100 to 2000) it can lead to many errors in tree reconstruction. This becomes even more problematic if distinctly non-linear transformations are being made in an attempt to eliminate inconsistency (i.e. ”corrected distances’). A reliable tree building method should be able to cope with these ’statistical errors’ in a competent way. If these were the only source of errors, the method which best deals with them is called ’statistically efficient.’

The second problem is that of multiple substitutions at a site at a site (i.e. multiple hits). As with parsimony, this factor can cause inconsistency of distance based methods. For example, multiple hits distort observed distances from additivity, because observed distances are progressively underestimated. An example of this is shown in figure 3. That is, the bigger the observed distance, the more (percentage-wise) it underestimates the true distance. It is ironic that while observed distances are expected to preserve all the metric properties (defined earlier in the parsimony section), they do not preserve additivity and this leads to inconsistency (as described later).

It is possible, after assuming a model, to make a transformation of the observed distances to, in expectation, correct for multiple hits and restore additivity. A useful transformation may restore distances close to additivity given sufficiently long sequences, and so in these instances is certainly desirable. A catch however, is that any transformation will increase sampling error’s (often a little, but sometimes a lot), and this can be most undesirable for short sequences where sampling errors may account for most of the problems in tree estimation. That is, a correction may be exact in expectation, but will introduce a larger degree of uncertainty, (since the transformation is extrapolating from known to unobserved data). Even worse, a transformations which is applied inappropriately (e.g. the data is i.r. but a proportion of invariant sites are removed) will not only increase the variance but head towards the long edges repel problem and the possibility of inconsistency in the anti-Felsenstein zone (Waddell, 1995).

Figure 3. A plot of the observed versus the true of total distance (as the expected number of substitutions per site). Notice how y, the deviation of the observed distance from the true distance (under a 2-state Poisson model, with identical site rates = i.r.) proportionally increases. If some sites are invariant (here pinv = 0.2), or if site rates follow a gamma (Γ) distribution with shape parameter $\alpha = 1$, there is even larger deviation (z). Notice how while the two curves allowing for unequal site rates are different, they are much more similar than either is to the observed or the i.r. curve (and even cross each other).

These then are potentially major problems facing distance based tree estimation. Four important ways of improving the performance of distance based tree inference methods aim to take these into account:

1. Using tree building method that best take into account stochastic or random errors in the estimated distances.

2. Transforming the observed distances to reduce systematic errors due to multiple hits (but at the expense of increasing random errors).

3. Using just those sites in the sequence that are expected to lead to reliable estimates of distance.
4. Detecting when distances are strongly distorted away from having additive properties, then reconsidering whether alternative transformations or data editing can help.

We will describe the first two factors in the next sections, while the latter two are considered near the end.

The description of factors related to tree selection methods rest upon general statistical knowledge, particularly of multi-variate models. The reader may wish to consult Felsenstein (1984, 1986, 1987, 1988), Bulmer (1994) and Swofford et al. (1996) for considerations in a similar vein.

16.7 Fitting imperfect distances to a tree: dealing with sampling error

Since we are always dealing with finite sequences, evolutionary distances are always expected to have sampling errors. Any single observed pairwise dissimilarity (dis) (i.e. the number of sites in an aligned pair of sequences showing different states) has a discrete nature, that is it will be $0, 1, 2, ..., c$ differences out of c total sites. Assuming that all site changes are independent of each other, the sampling distribution of dis will be binomial. For example, if the probability of seeing a change at any one site, p, (equals the expected value of dis divided by the sequence or $E[dis]/c$), is 0.2, and $c = 100$, then the observed dissimilarity in randomly evolved sequences is like sampling with replacement 100 balls from an urn with 20 blue balls and 80 red balls. Thus, the expected error on a single pairwise dissimilarity estimated from a random sample is binomially distributed, with variance $cp(1 - p) = c \bullet (E[dis]/c)(1 - E[dis]/c)$. Consequently, the sampled distance can be either larger or smaller than its expected value.

It is often convenient to consider the observed distance as $d_{obs} = E[dis]/c = p$. It is also important to distinguish a sample estimate of d_{obs}, from its expected value (the same applies to all statistical estimates). Thus, the observed distance d_{obs} is given a hat, i.e. $\hat{d}$, to indicate it comes from a sample (or more generally, any distance measure d is replaced with its sample estimate, $\hat{d}$). It is possible to also infer the variance of $\hat{d}$. Because d_{obs} is dis/c, we are effectively multiplying by $1/c$, so the variance of $\hat{d}$ is $1/c2$ the variance of dis, i.e. var[] = 1 / c (E[dis] / c) (1- E[dis] / c). Taking the example in the preceding paragraph, c = 100, $d_{obs} = 20/100 = 0.2$, so $var[d_{obs}] = 1/1000.2(1 - 0.2) = 0.0016$, thus, the standard deviation (s.d.) is = 0.04. Since a binomial distribution quite quickly begins to resemble a normal distribution when both $E[dis]$ and $(c - E[dis])$ are greater than 5, this observed distance is approximately normally distributed about 0.2 with s.d. 0.04.

Since we do not know dobs exactly, we usually estimate the variance of dobs using . For example, if the observed dissimilarity in our sequence is a little less than expected, say 17 rather than 20, = 0.17 and the variance estimated in this instance is 1/100 (0.17) (1-0.17) = 0.0014. This is still a fairly good estimate of the true variance, with its value positively correlated with the sampled distance. Another factor related to random error comes about because the pairwise distances are phylogenetically correlated. That is, when there is sampling error on one distance e.g. due to extra substitutions, another distance along a very similar path through the tree will likely be affected also. In contrast, a pair of distances that do not share any edges in the true tree in common are expected to have statistically independent errors. If we don't know what the true tree is, we require another method to directly estimate the correlation of distances. This turns out, like the variance estimate itself, to be very simple as long as we assume site changes are independent. The covariance of two observed distances $d_{obs(ij)}$ and $d_{obs(kl)}$ is given as,

$$cov[d_{obs(ij)}, d_{obs(kl)}] = [d_{obs(ij,kl)} - d_{obs(ij)}d_{obs(kl)}]/c,$$

where $d_{obs(ij,kl)}$ is the proportion of sites at which both sequence i is different to sequence j and sequence k is different to sequence l (with samples obs is used in place of d_{obs} to estimate covariance). This basic formula, which comes from general statistical theory of multinomial distributions, has been used implicitly or explicitly in variety of applications of distances to phylogenetics (Hasegawa et al., 1985; Nei and Jin, 1989; Bulmer, 1991; Waddell et al., 1994).

Consequently, if the two distances ij and kl are completely positively correlated (i.e., all differences between these pairs of species occur at exactly the same sites) this implies $d_{obs(ij,kl)} = d_{obs(ij)} = d_{obs(kl)}$. Conversely, if two distances are completely uncorrelated, then the expected number of sites where both i is different to j and k different to l is simply the chance of this many random substitutions being at the same sites, i.e. $d_{obs(ij)}$ $d_{obs(kl)}$. Clearly, then both variances and covariances are important in measuring how closely distances fit any given weighted tree. Their estimates are commonly put together in a single variance covariance matrix, **V**, covering all the pairwise distances being considered (with the variances being the diagonal entries). If variances and covariances are estimated from the data they yield an estimate of **V**, that is .

16.8 Least Squares Methods

16.8.1 The GLS criterion: a near ideal way to deal with sampling errors on distances

Generalized least squares (GLS) tree selection is perhaps the best way to deal with sampling errors on distances when fitting trees to distance data. While this method is not currently implemented in PAUP*, it holds the key to understanding the statistical properties of the other distance based methods which are available (including the FM criterion, minimum evolution or ME, and NJ described in detail below).

With site changes being independent, a matrix of distances becomes distributed as multi-variate normal as sequences get longer. This, plus being able to estimate the variance-covariance matrix, V, allows us to objectively consider various strategies for fitting trees to distances. Fitting a tree to distance data is closely related to a multivariate normal regression, where data points have unequal variances and are correlated amongst themselves. The ML estimator in these conditions converges to the method of Generalised Least Squares (GLS). When all distances are uncorrelated, i.e. all covariances are zero (not possible with sequences, but theoretically feasible with DNA hybridisation), the cross products of deviations from expectation are all zero and can be ignored. Then, GLS reduces to Weighted Least Squares (WLS). The optimal form of WLS with multivariate normal data measures deviations of the observed data (distances) from the expected distances dT (the path lengths through the tree being evaluated) as S(- dT)2 / var[], where the sum is over all t(t-1)/2 unique entries in the distance matrix. As soon as covariances become non-zero, they are used by GLS to take into account the correlation of a pair of distances. For example, if a pair of distances have a covariance of 0.08, and a high correlation (corr.) of 0.7 (the corr. of x and y is just, cov[x, y]/(s.d.[x]s.d.[x])), then in measuring the fit of a tree to the data, it is 'unfair' to let the deviations between tree and distance be counted twice (since they are practically the same piece of information). Instead, with GLS the measured misfit is discounted by their correlation (if these distances were totally correlated, i.e. 1 or -1, then their combined effect would count just once and not twice as it would if they were independent). GLS makes this same sort of adjustment for all pairs of distances simultaneously. This can be expressed in a compact algebraic form, since the covariances are the off-diagonal elements in a variance-covariance matrix, V. Thus, the GLS sum of squares (SS) can be written as

GLS SS = =

where x and y are the index given to the distances arranged in a vector for the observed values and dT for those pathlengths expected from the tree assumed. Further, as with standard GLS regressions, there is a closed form solution that minimises this SS, to give the optimal edge lengths (in vector) for a tree, i.e. = (XTV-1X)-1XTV-1. Here X is called an incidence matrix. It scores a 1 in entry mn, if edge m of the tree being considered contributes to distance n on that tree (else score a zero). Putting both of these equations together, the minimal SS of any tree topology can be evaluated as, SS = . A further manipulation gives the variance-covariance matrix of the edge lengths on that tree as (Agressti, 1990; Bulmer, 1991; Waddell, 1995). With long sequences, the GLS SS is a statistically complete criterion to measure the departure of observed distances from those expected if the tree generated the data. If the data perfectly fit a tree, then the SS goes to zero, and can only be zero on one tree (with additive data the true tree, and collapsing all zero length edges).

GLS is based on well known statistical properties, and allows a variety of statistical tests. Most specifically, if the model holds, the residual SS is expected to be distributed as c2 with degrees of freedom equal to t(t-1)/2 minus the number of parameters estimated in the model (e.g edge lengths, ti/tv ratios, distribution shape parameters, as described later). This then allows tests of fit of the data to model and whether any specific parameter (or combination of parameters) significantly improve fit of data to model. Indeed, practically all the tests made with ML on sequences can also be made with GLS on distances, using the SS just as one would use a G2 goodness of fit statistic. The main draw backs are that the GLS SS: becomes more expensive to calculate for large numbers of taxa than ML on sequences; does not appear to have as much power (differentiating ability) in its tests as ML on sequences (Waddell, 1995); and most practically, is not available in a computer package at present. However, while tests associated with ML on sequences which assume the asymptotic c2 distribution can be severely compromised by sparseness of the data [i.e., since many site of the 4t site patterns are rarely observed, e.g. Reeves (1992) and Goldman (1993)], distances sum many site patterns together and reduce this problem. This is a potentially great advantage that has not been explored.

In real instances, the covariance matrix V, is not known, but can be estimated in a number of different ways. One way is directly from the observed data (as described above), and results in the GLS method used by Bulmer (1991). An alternative strategy is that independently described by Felsenstein (1986), and described and used by Hasegawa et al. (1985). This method is analogous to the iterated GLS method common to many statistical packages. This uses the covariance matrix predicted by: the tree and the assumed mechanism of evolution (in one case the HKY 85 model, in the other the Jukes-Cantor equivalent for amino-acid substitution). Starting with the GLS tree, the original V is replaced by V2 predicted from it, then a new GLS tree estimated using V2, and so on (hence the name iterative least squares). Iterative least squares is well known in statistics as an ML method if the data is distributed as multi-variate normal (Stuart and Ord 1991) and V must be estimated. Accordingly, it has better expected performance with long sequences than GLS using estimated just once (Waddell, 1995), but has yet to be studied with simulations. Unfortunately, iterated GLS always costs more to compute than uniterated GLS, since each iteration (Waddell, 1995) involves the same optimisations as GLS (plus updating the expected covariance matrix V).

With small amounts of data a major weakness of GLS methods, with respect to other distance based methods, is that the estimation of V involves errors as large, or larger than those expected on the sample distance (especially in estimating covariances, e.g. Waddell et al. 1994). This compromises the performance of these methods, but decreases as sequences get longer. This is a directly comparable behaviour to ML which also may be outperformed by other methods with short sequences (Kuhner and Felsenstein, 1994). A second contributor is that short sequences will increase the deviation from a multi-variate normal distribution of the distances.

This description of the "ideal" distance based method (with respect to sampling errors) serves us as a framework to consider the properties of the more commonly used distance based criteria and algorithms, especially those available in PAUP*. The method of iterated GLS for distances may also be compared directly with that of ML on sequences, since the selection criteria are effectively the same. Analytic comparisons to date do suggest that fairly matched distance based methods lose both statistical efficiency and robustness with respect to ML on sequences (Waddell 1995, cf. earlier studies e.g. Hasegawa and Fujiwara 1993, Yang 1994 which compare unmatched criteria). Since optimality criteria were matched (i.e. ML on sequences versus ML on distances) the difference must be a consequence of the restriction the form of the original data (i.e. t(t-1)/2 unique distances versus xt possible character patterns, where x is the number of character states, e.g. Penny 1982). Consequently, it is important to understand the statistical assumptions of distance ased methods, if we are to get the most out of them.

16.9 Weighted least squares, FM method, minimum evolution, and neighbor joining

PAUP* presently offers these three criteria which use the additivity of distances to infer the optimal tree. We now describe their properties in relation to the statistical ideal of iterated GLS.

16.9.1 WLS and the Fitch and Margoliash (1967) (FM) criterion

If we ignore the covariances of distances, the weighted least squares (WLS) criterion,

WLS SS = (*)

Using weight, wx = s2x (the variance of dx) is statistically optimal for WLS applied to multivariate normal data, and this criterion has been employed in searching trees by De Soete (1983) and Olsen (1987). The

estimated variances of most transformed distances can be readily obtained via the delta method, as described later. A linear algebra solution to WLS fitting of sampled distances to a tree can be obtained in a similar manner to that for GLS (Swofford et al., 1996). The FM criterion is a simplification of WLS. It does not calculate the variance of distances explicitly, rather it assumes that they always increase as the square of the distance (i.e. if distance doubles then variance increases four times). Thus, in the FM method wx = 1/(x)2. This expectation has been little explored in comparison to the true behaviour of variances with different distances. Later we show that in some cases it can be quite realistic.

Unfortunately, while the WLS fit statistic is a sum of squares, its distribution is not c2. The true distribution can be calculated with intensive calculations (see Bulmer 1991). Further, the FM methods SS, because it is measured proportionally (but not absolutely) cannot be used to compare between data sets or transformations of the same data in any especially meaningful way (as it does not take into account the absolute size of variances). Accordingly, the FM SS should be used very cautiously in making any comments about the fit of data to model.

WLS, including the FM criterion, can be estimated more quickly than GLS using techniques that take advantage of the fact the covariance matrix has all off-diagonal entries set to zero. Analogous to GLS, an iterated WLS method is also be possible (here s2 would be estimated from the tree path lengths, dT, on each cycle), with each iteration necessary for convergence to the optima adding to the computational expense. While WLS methods are slower than the some other distance methods, they have a better understood theoretical foundation and way of dealing with sampling error. Even though the FM criterion does not use the optimal weighting, simulations by Kuhner and Felsenstein (1994) show it performing at least as well as minimum evolution on the OLS tree, or Neighbor joining. This result is at odds with some earlier studies, which did not take the sensible step of constraining all internal edges in the tree to be non-negative (discussed further below). We define FM constrained in this way as FM+. While, GLS and WLS were not simulated, by Kuhner and Felsenstein (1994), our own unpublished simulations (Waddell and Reeves, in preparation) suggest they are better again as long as the constraint is imposed.

If all distances are weighted equally (i.e. weight wx is set to 1, or the observed distance raised to the power zero), this gives the ordinary least squares (OLS) criterion (e.g. as used for standard regression lines, and for trees by Cavalli-Sforza and Edwards 1967). This weighting assumes that all distances have the same sized sampling errors. Felsenstein (1984, 1993), gives a generalisation of tree fitting, by allowing the weight in equation * to be proportional to any power of the observed distance. This allows some fine tuning to better match the expected sampling errors of non-sequence data, but does little to allow a better approximation to 1/s2 with sequences (as explained later).

This then, covers the progression from the most sophisticated to the least sophisticated least squares methods. The progression of allowing more statistical information into tree selection is unfortunately paralleled by increasing computational complexity. For a discussion of this matter see (Waddell and Bryant 1996).

16.10 Minimum Evolution

Minimum Evolution criteria are generated by combining any optimality criterion that gives edge lengths (criterion x), but taking as the score of that tree as its sum of edge lengths. The idea is, to then search over all trees to find the tree with the minimum sum of edge lengths, or the minimum evolution (ME) tree. For distance methods, common methods of estimating edge lengths are by least squares. Examples are OLS, and OLS+ where the constraint that all edges must be positive is applied. Such methods were applied by Kidd and Sgaramella-Zonta (1971) and Rzhetsky and Nei (1992). Saitou and Imanishi (1989) applied minimum evolution to the FM criteria, while Rzhetsky and Nei (1993) considered its application to WLS and GLS (unfortunately all without the constraint of forcing edges to be non-negative). Interestingly, if you estimate edge lengths by parsimony (weighted or not), then apply ME, the trees score remains the parsimony score. In other words ME(parsimony) reduces to parsimony. The term minimum evolution was originally used as an acronym for maximum parsimony and its use in conjunction with a distance method has caused concern (Edwards 1996). However, this is clearly misplaced, as long as it is made clear what the ME criterion is being applied to. Thus we use a two part description, such as ME(OLS), or ME(parsimony).

Recently, ME has also been applied to trees with edge lengths estimated by maximum likelihood (Waddell 1995, Adachi and Hasegawa 1996). In Waddell (1995) the method had a very high correlation with the ML lnL score. However, in Adachi and Hasegawa (1996) the method applied to a sample of the same hominoid mtDNA sequences, gave a different tree to ML and apparently an incorrect tree (since human and chimp are expected to be together). The reason appeared to be a volatile transition to transversion ratio, so if transitions (or any other part of a model) are near saturation and so, can cause fluctuate an associated parameter to fluctuate wildly, it may be wise to restrict ME(ML) to the more conservative changes or else fix the relevant parameters. In the case mentioned, this gives the expected tree. However, ME(ML) also shows a surprising turn of robustness. If you take the example data used by Adachi and Hasegawa, and apply JC ML you get an incorrect tree (grouping human and chimp), yet ME(JC-ML) identifies the correct tree (lnL scores 617.49 vs 617.95, ME scores 1.438 vs 1.429, respectively), hinting at it possibly being a robust estimator in certain circumstances. ME(ML) can easily be evaluated in PAUP* by describing the best 50 trees previously found by an ML search (clicking the branch length box), then summing up the edge lengths.

Why ME(x) should generally work is uncertain. However, an analogy to parsimony, and likelihood may provide an insight. When the tree is not the optimal likelihood tree, the external edges especially become longer, while any internal edge not corresponding to an edge in the true tree will tend to shrink towards zero (Waddell, 1995). The balance is that the external edges grow more quickly in size than the internal edges decrease. This behaviour is understandable, since each tree model must do its best to "predict" (maximise the likelihood of) the observed data. Thus, if these are an lot of patterns grouping specific taxa (external edges) together then the ideal solution is usually to explain these arising on an internal edge. When the tree being optimised can not have this edge, the best it can do to "predict" the patterns it sees is to make the external edges which need to be grouped longer, so that they will imply the build up of multiple hit patterns corresponding to the patterns strongly grouping edges. In general, it takes a larger increase in external edges, then it does a length of internal edge, to generate a specific pattern at similar likelihood since it is the result of a series of more unlikely events (parallelisms or convergences). A similar thing seems to happen with distances, although it is harder to express directly as pattern likelihoods. This suggests that an increasing sum of edge lengths is a consequence of a badly fitting tree, by either criteria. We return to this feature when we discuss the question of whether to allow negative internal edge lengths or not.

In conclusion then, MSE appears to be an emergent property of any consistent tree optimisation criteria, in that it costs more in edge lengths to explain the same state in unrelated taxa as due to separate origins, than it does to explain them by common ancestor. This then raises interesting questions like "when is it better to trust this emergent quality over the explicit criteria (e.g. likelihood)?" Until this question can be answered generally, it seems likely that ME methods will not fit into the statistical frame work of the best understood model based methods (e.g. GLS, WLS, OLS, or ML on sequences). It appears that some consensus is beginning to emerge for ME(OLS) as we discuss next.

16.10.1 The minimum sum of edge lengths on the OLS tree, ME(OLS)

This criterion uses (OLS) to get the edge lengths for a tree, but the score of that tree is then the sum of the edge lengths (Swofford et al., 1996). The smaller the sum of edge lengths, the more optimal the tree is. Thus, the minimum OLS tree need not be the optimal tree by the OLS criteria. This criterion was first developed and tested with simulations by (Kidd and Sgaramella-Zonta 1971). More recently, it has been resurrected by Rzhetsky and Nei (1992), apparently unaware of the previous work, and been renamed minimum evolution (ME).

There are variations on how the sum of edge lengths is taken, and whether the OLS optimisation should be allowed negative edge lengths. Kidd and Sgaramella-Zonata (1971) favoured no constraint on the OLS optimisation (i.e. negative edge weights were allowed) but favoured the sum being of the absolute values of the edges (we call this MEabs(OLS)). This means a negative edge weight is effectively penalized. Rzhetsky and Nei (1992) also favoured the unconstrained OLS optimisation, but a straight sum of edge lengths, so that a negative edge length could reduce the overall sum (method ME (OLS)). It is also be possible to apply ME to an OLS tree constrained to have all edge lengths non-negative (we call this ME (OLS+) = MEabs (OLS+)). The question of the meaning and treatment of negative edge lengths is considered in detail later.

Simulations so far, have tended to indicate that all ME (OLS) variants may be superior to straight OLS (without the constraint of positive edge lengths, Kidd et al. 1974, Rzhetsky and Nei 1992), and about the same as the FM method when any tree with a negative internal edge length was automatically rejected (Kidd et al. 1974). In contrast, Kuhner and Felsenstein (1994) indicated poor performance by ME (OLS) compared to either FM+ (MEabs (OLS) or ME (OLS+) were not considered). Theoretical studies by Rzhetsky and Nei (1992) predicted ME (OLS) and ME (GLS) (i.e. the minimum sum of edge lengths on the uniterated GLS tree) outperforming OLS, WLS and GLS. However, only four taxa were considered and all methods were allowed to reconstruct negative internal edge lengths.

Evaluations of least squares methods and ME methods (Waddell and Reeves unpublished) are however suggesting that Rzhetsky and Nei's (1992) result is biased, so that GLS with the constraint of a positive internal edge does better than ME(OLS). This is consistent with the suggestion in Swofford et al. (1996) that ME(OLS) may only outshine least squares methods when negative edges are allowed. Overall, these studies (Kuhner and Felsenstein, 1994) are suggesting that ME(least squares) works better than least square methods when negative edges are allowed, but like other distance methods its performance also improves by forcing edges to be non-zero (but not as much as OLS, WLS or GLS improve). Still uncertain is which variant of ME will be most robust when the data does not fit the tree well, and whether ME(least squares) has any clear robustness properties over least squares.

On the plus side, MSE (OLS) is faster than FM to optimise per tree (although often 10 or more times slower than unweighted parsimony), and this facilitates more thorough searches. Overall then, MSE(OLS) criterion is an interesting method, with the best performance under the model expected as MEabs or ME(OLS+), but with unstudied robustness properties. If any variant of ME(OLS) does turn out to be as reliable as the FM+ method, then this will be desirable as it is faster in its implementation.

16.11 Algorithmic Methods

16.11.1 The Neighbor-joining algorithm

Unlike the previous methods, this is not a criteria but an algorithm to 'build a tree'. In this sense it is similar to other distance based algorithms such as distance Wagner (Farris 1972), the Sattath and Tversky (1977) (ST) method, and transformed distances (TD) (Li 1985). Of these methods, all but TD (Charleston 1994, Charleston et al. 1993) are consistent with additive data (i.e. they do not assume equal evolutionary rates in all lineages). Neighbor-joining was first described by Fitch (1981), who called it 'gradual neighborliness' (Gascuel 1994). Apparently unaware of this, Saitou and Nei (1987), redescribed the criteria and an order t5 algorithm calling it 'neighbor joining.' However, the algorithm most commonly used to stepwise maximise 'gradual neighborliness' is the order t3 'neighbor joining' algorithm of Studier and Keppler (1988, see also Gascuel 1994). A simple step by step description of NJ is given in Swofford et al. (1996).

Simulation studies of the relative performance of these various algorithms have been mixed, incomplete or inconclusive in the comparisons made. Probably the most complete study is by Charleston (1994) (partly presented in Charleston et al. 1994). Here, it was found that ST and NJ performed nearly equally well and had some tendency to choose similar trees, and clearly performed better than TD. Overall, the

Studier and Keppler NJ algorithm would often be the method of choice as it is order (t3) while ST is order (t5), making bootstrapping noticeably slower (Distance Wagner method was not compared then, but has been by now by Waddell et al. 1997, and consistent with earlier studies (e.g. ?) it does not perform as well). Overall, the results suggest that NJ, while performing acceptably, was not clearly superior to another clustering method (ST) based on the stepwise maximization of discrete neighborliness (i.e. that each new edge added to the tree, was favoured as it gave the highest number of sets of four taxa obeying the Buneman 4 point metric of figure 1).

Overall, NJ does quite well in most studies. In Kuhner and Felsenstein (1994) it did not perform as well as ML, but did perform nearly as well as FM+ at identifying the tree (although worse at estimating the edge lengths). In Charleston (1994) and Charleston et al. (1994) NJ was one of the better performing methods, but clearly lagged behind parsimony. Other studies have suggested reasonable performances, although those suggesting it has the best performance have since been shown to be inaccurate (Swofford et al., 1996).

NJ itself has a tenuous relationship to ME(OLS). As Rzhetsky and Nei (1992) note both methods always favour the same tree with four taxa (but so to does ST and at least three other methods, which all reduce to choosing the tree (i,j)(k,l) when dij + dkl is minimal, see figure 1). With more than four taxa, when NJ adds an edge to a tree, this corresponds to the split amongst the remaining unresolved taxa which will approximately minimise the ME (OLS) criterion. Extrapolating one step further, a loose claim is that NJ behaves most similarly (of described optimality methods) to a star decomposition search algorithm with the ME (OLS) criterion. However, why NJ does quite well in simulations is not clear: it has a different criteria to ST but does very similarly, while its implicit search strategy, star decomposition (Swofford et al., 1996) does not perform especially well with either parsimony (DLS unpublished) or ML as the number of taxa are increased (Adachi and Hasegawa 1996). Thus, there is no good evidence to suggest NJ is a superior method due to its loose association with ME(OLS). We mention these facts to counter unsubstantiated claims which infer that NJ is somehow the best tree building method (e.g. Nei 1991, Kumar et al. 1993), a claim for which there is no good empirical or theoretical evidence.

Another claim associated with NJ which has no justification is that NJ is within one or two nearest neighbor rearrangements of the ME tree (e.g.

Rzhetsky and Nei 1992, Kumar et al. 1996). When doing ME searches in PAUP* for more than a few taxa (e.g. 20 plus) it will be very obvious using PAUP* that the ME(OLS) tree found is often quite a few partitions different to the NJ tree. As most people would expect, this gap between the NJ tree and the ME(OLS) tree tends to become wider the more taxa are used. Often, it may be that the NJ tree will be unable to be differentiated from the ME(OLS) tree by statistical criteria, although this need not be the case in all instances (discussed further below).

Despite the exaggerated claims for the methods accuracy, the great speed of NJ offers benefits. NJ is a method which allows quick bootstrapping of large trees, while still allowing model based transformations of the data. In speed it is clearly faster than ME (OLS) even with a local search strategy, since ME(OLS) requires iterative optimisation of the OLS branch lengths, whereas NJ does not (it makes a best first guess). The major disadvantage is that of any tree estimation which does not consider rearrangements after estimating the tree: getting stuck in a local optimum. With 40 taxa, if even one of the later splits is got wrong, it could potentially see 10 or 15 partitions in the whole tree wrong. This may be rare when the data fits the model, but of greater concern when it does not. On the other hand, when data does not fit the model, it may be that a search strategy like star decomposition does well in some instances. Since users are going to want to bootstrap large trees, a wise move might be to do the NJ bootstrap analysis, but check that the optimal ME (OLS) tree (and hopefully even a handful of bootstrap replicates) tend to agree with the NJ tree on the features of most interest.

16.11.2 The unweighted pair group method (using arithmetic averages): UPGMA

UPGMA is a simple algorithmic clustering method that makes the often strong assumption that the data are ultrametric (i.e. completely clock-like). It works on the simple premise that if there is a perfect clock, then those taxa which are most closely related will also have the least genetic distance between them. Thus with just four taxa, the estimated tree will be determined which of the 6 distances is minimal, and join this pair of taxa first.

Violation of the assumption of a perfect-clock can cause UPGMA to be inconsistent. Indeed, its susceptibility to inconsistency due to violation of this principle, seems to be much larger than that of other methods which are also susceptible to unequal evolutionary rates (e.g. parsimony, or compatibility). With short internal edges, it can be very difficult to detect where UPGMA may be going wrong, especially if the data are near ultrametric. Additionally, because of the methods susceptibility to systematic errors due to unequal rates, this methods bootstrap support for the branching order can also quickly become very distorted (so that an edge which is incorrect due to systematic error caused by unequal rates can quickly show 100% bootstrap support!).

Perhaps even more importantly, UPGMA does not appear to be as statistically efficient at estimating the tree topology as NJ, even when the data are perfectly clock-like in expectation. A number of simulations have hinted at this possibility, but part of the problem is that UPGMA can be more efficient than NJ on certain types of tree (especially trees which are somewhat ladderized, unpublished observation). The simulations which seem to put this result beyond doubt were conducted in Charleston (1994) and a follow up Waddell et al. (1996). For each simulation 1000 trees of up to 50 taxa, where generated, obeying the molecular clock, but with random branch lengths. In all cases NJ did noticeably better than UPGMA (e.g. Charleston 1994, fig. 6.1-6.4). (Note however that parsimony was tending to be markedly superior to NJ if rates were not high, Charleston et al. 1994, Waddell et al. 1996). UPGMA may retain interest for those who wish to estimate approximate divergence times from distance data, although a clock-constrained least squares fit would be desirable (e.g. available in Phylip 3.5, Felsenstein 1993).

16.12 The treatment of negative edge lengths

Trees reconstructed from distances can have negative internal edge lengths. To understand how to best treat them, it helps to understand what causes them. If you draw a tree, and give it both positive and negative internal edges, it will still define (and be uniquely defined by) a matrix of additive distances, although some of these pathlengths may be negative (i.e. when the negative edges make up over half a path length through the tree, Hendy 1991). This is purely a mathematical definition, but in it negative edges have a clear interpretation: they say ”subtract the length of this edge form all paths that cross it.” In reality, for Markov models of evolution, the tree has all edges non-zero, and all distances non-zero. The problem comes when dealing with data which is not additive on the 'true' tree (or perhaps any tree), which may be caused by sampling errors and / or systematic errors (i.e. not properly accounting for multiple hits). In these situations, the distances will be all positive, but internal edges may still be negative on certain trees, for certain optimisation criteria. This is simply because the matrix of distances is closer to additive on a tree with some negative edge lengths. In these instances, the negative edges often have a fairly direct interpretation.

Let's assume there is just one negative internal edge in the tree, separating the group of taxa A from the group of taxa B. This clearly indicates that the distances within group A are most additive upon a subtree with all positve edge lengths (with some errors), as are the distances within group B, but when you start to measure distances from group A to group B they are generally biased to being too short (so putting a negative edge length between the two groups indicates this, without needing to distort the subtrees for groups A and B). This being the case, the edge is not interpretable in the same way as a positive edge; it is quite literally an 'anti-edge', so that instead of indicating a separation of taxa it is indicating some unexplained attraction between taxa in the different groups, A and B. As such its most logical interpretation is that there is no evidence to separate the taxa concerned in the way indicated, i.e. there is less than no evidence for this edge in the tree. Consequently, our best guess of the magnitude of this edge in the true tree is zero (since we accept the true tree has no negative edges). Indeed, it is exactly this same interpretation which is applied when sequence data or distances are corrected using Hadamard transformations (Hendy and Penny, 1993; Waddell et al., 1994; Swofford et al., 1996).

The above interpretation is our own, but Felsenstein (1984, 1986, 1988) and Farris (1985) consider the same issue from distinct perspectives. Felsenstein pointing out that negative edges do not correspond to the model, and so using prior information should not be allowed. Farris (1985), seemingly playing devils advocate, says they are an unavoidable consequence of trying to fit imperfect additive data. To paraphrase Farris, "if you believe the model and trust the optimality criterion you must accept negative edges." Our interpretation is clearly much closer to Felsenstein's interpretation, yet the acid test must be which interpretation does best in practice, i.e. in real instances. That is, does rejecting a negative edge length, on average, help to better resolve the tree. Kuhner and Felsenstein (1994), support this view with simulations. Alternatively, if you take the results in Waddell et al. (1994) and choose the tree with the least negative internal edge (preferably with weighting by one over the standard deviation of that edge), calculus shows this maximises the probability of the correct tree (when choosing each edge without consideration for the size of any other edges).

Searching amongst trees with the constraint of forcing all edges to be non-negative edge lengths (the + constraint). From theory, it follows that with non-iterated least squares optimisations, there is but one minimum for an edge with a length allowed to be any real number (positive or negative) (and this includes when allowing all other edges to be simultaneously reoptimised). Accordingly, if an edge is negative, it follows that the non-negative value for this edge which will see the least deterioration of the SS (best fit) will be zero (Bryant and Waddell 1996). Such a tree, is also an edge length optimised polytomous tree. This result also implies that any tree that differs by rearrangement of just this negative edge (i.e. an nearest neighbor interchange about this edge), must have a smaller SS than the polytomous tree if this edge when optimised takes on a positive value. Contrary to Swofford et al. (1996), the solution to the minimum sum of squares for a polytomous tree is easily obtained from the standard algebraic solutions (Swofford et al., 1996, eq 14, 15) by supplying the incidence matrix of a polytomous tree (see earlier section).

Thus, the score of any tree with a single negative edge can be equated to that of the polytomous tree with this edge collapsed. Further, it need not be evaluated explicitly if searching for the best tree (i.e. it can be automatically rejected), unless there is no positve edge resolution amongst the four sets of taxa incident at this edge. Thus, only if all three rearrangements are negative, does this fail to resolve an internal edge. Here the SS of the three trees will all be set to that of the optimised polytomous tree, however, if you were forced to guess which of the three resolutions was most likely (i.e. which came closest to a tree with a positve internal edge), it would be that with the optimised edge which is least negative (Waddell, 1995). This general rule can be applied for OLS, FM, WLS and GLS with the + constraint. This treatment is slightly different to that presented else where (Swofford et al., 1996), where it is suggested the negative edge be set to zero without reoptimisation of edge lengths (this will occasionally result in the optimal arrangement (i.e. optimal tree) being bypassed in the search). With more than one edge in the unconstrained least-squares tree being negative, imposing a constraint of all edges non-zero, will see one of these edges take on zero value, while the other may be zero or positive. Here, collapse of negative edges to yield a polytomous tree (or rejecting the tree topology outright) could also see the optimal tree missed by the search. For appropriate constrained optimisation algorithms for such cases (or with nay number of negative internal edges) see Bryant and Waddell (1996).

In the case of ME methods, things become more difficult as there is a hybrid selection criterion in force, and because PAUP* offers many permutations on the theme. Each alternative can be argued to be valid, with some already having proponents. PAUP* offers the user the options ME(OLS) (Rzhetsky and Nei 1991), ME(OLS+) and MEabs(OLS) (Kidd and Sgaramella-Zonta 1971). Relative to each other, the first criterion is expected to most favour an OLS tree with a negative internal edge. The third criteria will penalize an OLS tree with a negative internal edge by turning it against that trees optimality (i.e. by taking the sum of absolute values). However, it is the second criterion which may inflict the most penalty of the three on an OLS tree favouring a negative internal edge, by forcing the edges to be more ill adjusted, and thus potentially most increasing their total length. (Note, how PAUP* achieves the OLS+ constraint is described in section following the next). Given the theoretical uncertainties over this method, and the lack of any simulations comparing these alternatives, in practice the user may well wish to explore these three different options and report any case where they make a substantial difference (e.g. bootstrap support shifting by 30% or more on edges of a priori interest).

The NJ algorithm may also be implemented in a number of ways so as to avoid negative edge lengths. The modification used by Kuhner and Felsenstein (1994) does not affect the tree selected by the algorithm. Rather but each time a negative edge is indicated, it is simply set to zero by readjusting the lengths of the adjacent branch.

David: Can we ask the question using NJ, which of the three alternatives of a clustering gives the edge length which is least negative when using NJ? I imagine not, since a star decomposition is in progress. Rather, we may be able to make NJ more like a constrained ME(OLS) search by forcing it to reject any clustering which leads to a negative edge length. That is when the algorithm ”realizes” it is reconstructing an edge with a negative length, it undoes the cluster concerned and takes the second best cluster in the table of rate-corrected pairwise distances, etc.

Lastly, it was suggested by Farris (1985) that a negative internal edge indicates that the model is being violated. Felsenstein (1986) quite rightly pointed out, that this need not be the case as the error must be larger than that expected randomly. We propose a simple test of the hypothesis that an internal edge is too negative to be due to chance. Bootstrap the original data n times using an optimality criterion that allows negative internal edges to be reconstructed. Then count how many times out of n the optimal tree found for each replicate has a negative value on the negative edge in the original data (let this number be x). If x/n is greater than 1-a (where a is the probability of a type I error, i.e. rejecting the hypothesis that the result is due to chance, when it is) consider there to be evidence for a significant violation of the model. Unless you had a priori suspected this internal edge of being negative, there should be a correction of the significance level for implicit multiple tests (i.e. how many edges might have had a negative value). Unfortunately such corrections are presently very difficult to make accurately, but this will act to lower the significance level, while the conservatism of the bootstrap will be acting to increase the significance level. Overall x/n of greater than 0.95 may be unlikely enough to warrant mention. A similar test can be made with delta method estimates of edge length variance e.g. Li (1989), Bulmer (1991), but they too have similar complications when estimating the significance level.

Interestingly, in cases of both long edges attract and long edges repel, some distance optimality criteria (e.g. ordinary least squares) may infer negative edge lengths. Indeed, allowing negative edges can maintain consistency when long edges are being repelled (Waddell, 1995), but further study is needed before making recommendations on how this might be used to diagnose the model.

David, is this worth mentioning somewhere:

Farris (1981) and (1985) made a big point of insisting that all methods of building trees from distances, force all observed distances to met the metric property. Felsenstein (1984, 1986) answered this claim by pointing out that genetic distances behave more like expected values, and can be both larger or smaller than the actual distance on the tree. A good analogy to the two approaches is a simple x-y regression. Farris was claiming that all distances must be underestimates of the true distance (which is in turn estimated from the pathlengths through the tree being reconstructed). If one were doing an x-y regression this might be taken as the regression line (analogous to the tree) must pass above all data points, irrespective of their location. Felsenstein (1982, 1984, 1986, 1987) clearly favours the expected value approach which says that our transformed distances should be additive in expectation, but can have error in either direction. Thus it is like a regression where all data points are assumed to potentially be either too high or too low, and a least squares fit is then optimal if we assume the data points have approximately normally distributed errors. There is however one instance where insisting on the metric property may be useful, and that is when using observed distances, but believing the model is in fact subject to multiple hits. Thus the observed distances must logically be less than or equal to pathlengths through the tree. Whether this constraint can actually help a method is, in practice, is unknown.

16.13 Edge length estimation

Distance based methods can be used to estimate the lengths of edges on a tree of methods such as relative divergence time estimates. The tree that edge length estimates are supplied could be constructed with any method (e.g. if the maximum parsimony method was considered most reliable with the data set in question, then it might supply the topology). This is unlike UPGMA which gives local optimisation. The statistical efficiency of edge length estimation is expected to follow an equivalent trend to that for tree selection, i.e. iterated GLS, then GLS, then WLS, then OLS (ME methods are used to select tree topology, but not estimated edge lengths). Hasegawa et al. (1985, 1987, 1990) appear to have appreciated this fact, in that all their molecular clock constrained trees for divergence time calibrations used iterated GLS. Using uniterated GLS with a clock constraint on the tree (e.g. Bulmer 1991) gives a method which is not quite as statistically efficient. Interestingly, the FM+ method tends to have a slight advantage over the NJ+ method for estimating edge lengths (Kuhner and Felsenstein, 1994), although this could be due to either its use of a non-optimal form of WLS, its explicit optimisation, or its ability to search out trees slightly closer on average to the true tree. It is expected from the tendency of the variance of a sequence based distance to rise at least as (p(1-p)), that FM+ optimization will outperform OLS+ in edge length estimation most of the time (however, the difference will fluctuate depending upon the pattern of the variances of the distances being used).

However, it is now practical using PAUP* to estimate tree edge lengths using ML with a much wider variety of models than GLS or iterated GLS, and the option of the clock constraint. Theory (e.g. Felsenstein 1981), simulations (Kuhner and Felsenstein, 1994) and applications of each method to real and analytic data (e.g. Waddell 1995) all suggest that ML on sequence patterns will have smaller sampling errors than GLS on distances (except possibly with short sequences), and even more importantly in many real applications, improved robustness when the model is incorrect. Further, with short sequences, ML takes into account the exact expected multinomial sampling errors, while GLS makes the perhaps unjustified assumption that they are close enough to multi-variate normal as not to mater. Taken all together, these findings suggest that usually the main time you might wish to use distances to estimate relative divergence times would be when the LogDet / invariant sites model seems most appropriate (since it is a very general and distinct model not implemented for ML as yet, see below).

16.14 Optimization routines in PAUP* for FM, OLS and ME (OLS)

For the all the non-alternating (e.g. variance estimates are updated as the edge lengths change) least squares methods (GLS, WLS, FM, OLS) there are closed-form linear solutions to find the optimal edge weights. Expressed as matrix and vector multiplications these are simple expressions. For example, with GLS the optimal edge lengths , are given by = (XTV-1X)-1XTV-1d, so substituting the tree pathlengths in as the minimum GLS SS = dTV-1d - XTV-1d for that tree (here X is an incidence matrix, coding for which edges occur on each path through the tree, T denotes transpose and d is a vector of the observed distances). For WLS the optimal edge weights are = (XTW-1X)-1XTW-1d (where W is the diagonal matrix of weights, so the same for FM but with different weights) while for OLS they are even simpler = (XTX)-1XTd (i.e. the weighting in V effectively disappears as all weights are one, with no covariances). These formulae are well known in multivariate regressions (e.g. Agresti 1990, and examples of their use in trees include Cavalli-Sforza and Edwards 1967, Kidd and Sgaramella-Zonta 1971, Olsen 1988, Bulmer 1991).

In practice these matrix algebra solutions are not the most computationally efficient, especially as the number of taxa increases (particularly when taking the inverse of V). To avoid this PAUP* uses numerical iterative refinement techniques. These are ? (please describe David). Further, since convergence is typically initially rapid, after a couple of iterations if a tree does not have a SS close to the optimal tree it seems safe to assume that the improvement in its SS due to further iterations will not bridge the gap. Taking advantage of this, PAUP* offers a settings box where the user can specify after how many iterations the fit will be measured, and how much worse than that of the optimal tree it must be before that tree is discarded. This method appears to work very well in practice, and a safe set of setting appear to be ... (which are the program defaults). Note: a similar option appears for ML which allows a search for trees using the approximate likelihood (e.g. after 5 iterations).

David, should we make recommendations on these setting with respect to the number of taxa be analysed?

16.15 Transforming distances to improve additivity

The usefulness of transforming distances, as mentioned earlier, is to make them more additive in expectation. In order to make a transformation it is necessary to assume a model, then using observed features of the pairwise comparison, extrapolate to predict what the total distance would be. In the course of making these extrapolations, the hope is to make the distances more additive in expectation, by accounting for multiple hits.

With finite data, the extrapolation to predict the total distance is particularly affected by sampling errors in that frequencies of the original observed data. The more parameters the assumed model has, generally the larger the variance the extrapolated distance will have. This is often due to the distances based on models with more parameters making more extreme extrapolations (i.e. larger corrections). As such it is theoretically desirable to use a distance which offers good improvements in additivity for minimal increase in variance. In practice, it is often crucial to consider most seriously the results of tree estimation after correcting the data with a transformation which accounts for the largest number of major features of the data. This may not yield the method which has the highest probability of inferring the true tree (i.e. if we could rerun the process over and over), but it will hopefully yield the best estimates of support (e.g. by the bootstrap). We consider further the conflicts in choosing a distance correction in a later section. PAUP* contains a wide range of distance based transformations for DNA and RNA sequences and a smaller number for amino-acid sequences. Those for DNA fall into two main classes. The first class, are distances based on time reversible models (this term is explained later), the second are LogDet distances which are based on a significantly more general i.i.d. model. Both classes are also be modified to allow for unequal rates of evolution at different sites when this is a prevalent feature, as it is with most functional DNA sequences. Recently developed distance transformations allow corrections to be made in PAUP* under any time reversible model with a variety of assumed site rate distributions. Described here in detail are the transformations used with 4 states, but they easily generalise to any number of states.

16.16 Some basic terminology

All the distance transformations used in PAUP* are based upon a transformation of the proportions of the paired nucleotide frequencies. That is, for the pair of aligned sequences i and j what proportion of the time does i have A and j have an A at the same position, i and A and j a C, etc. The proportion of ”AA” is just its frequency in the aligned sequences divided by the sequence length (or more specifically the number of sites where the comparison is being legitimately made, e.g. excluding sites with gaps). This information is conveniently stored in the divergence matrix F, which has 16 entries with 4-state sequences (or 202 = 400 entries with aminoacids). Thus, all F matrices will have entries which sum to one. The aligned pairs of sites, e.g. pattern ”AG” can be called either ”dinucleotide frequencies” or less ambiguously (since dinucleotides are also commonly referred to in moving along a single sequence) ”paired nucleotide frequencies” or just ”FAG” if they are normalised proportions (i.e. divided by the sequence length).

The frequency of each state (e.g. of A, C, G, and T) is also an important term in many distance transformations. These frequencies are usually expressed as proportions (so they sum to 1) and are often written as a vector, e.g. p = [0.1, 0.2, 0.3, 0.4], or a diagonal matrix P (a matrix with all entries zero, except the ii-th entries along the diagonal which are the same as the i-th entry of p). It is often required to differentiate exact model generated data from samples (e.g. real sequences), so hats are again used when this distinction is important (e.g. p hat, are base frequency estimates from a sample).

16.17 Distances estimates based on time reversible i.i.d. models

A time reversible model is one where the position of the root on the tree makes no difference to the probability of the data under that model (i.e. its likelihood).

A consequence of assuming reversibility is that the F matrix for each pair of sequences is expected to be symmetric. Other major characteristics of a time reversible model are that there is usually but a single underlying rate matrix R (see chapter on ML tree estimates), and the base composition (p) at the root of the tree (and so everywhere else on it) are the equilibrium values for R (these are the values you would get by squaring R over and over again, as if the process had been in operation for a very long time, which implies vector pR = [0, 0, 0, 0]). This means base frequencies are not expected to change through time or in any lineage.

Under such a stationary model, the total number of substitutions is just the product of the initial bases frequencies with the respective rates, multiplied by the overall time, i.e. , given i j, and t is proportional to time if a molecular clock is assumed, else it is just a scalar determining the total amount of evolution. When working with matrices, this amount is just -trace(PRt), since the diagonal elements of R are negative the sum of the row elements.

Given a first order Markov process, a reversible rate matrix and time, it is easy to predict what F will be if P is stationary,

F = PP = Pexp(Rt), (*)

where P is a transition matrix (elements being the probability of a site being in state i initially then being in state j at the end of the process), and exp is the matrix exponential function. The matrix exponential is defined as exp(Rt) = = I + (Rt)/1! + (Rt)2/2! +, ... , + (Rt)n/n! + ... Treating time as being fine grained (for example 1000 equal spaced intervals of change), then,

exp(Rt) (I + Rt/1000)1000 = (I + Rt/1000)1 (I + Rt/1000)2 ..... (I + Rt/1000)1000,

where I is the identity matrix (1's on the diagonal, 0 elsewhere). Every member of the series (I + Rt/1000) is the transition matrix (probabilities of change) in a short interval of time (t/1000). This relationship becomes exact as Rt is divided up an infinite number of times. Conveniently, (Rt) can always be diagonalised (a special property of rate matrices) so exp(Rt) = Wexp(Y)W-1, where W is a matrix of the right eigenvectors of (Rt), W-1 is its inverse, Y is a diagonal matrix of the eigenvalues of (Rt), and the exponent function is applied to each diagonal element of Y in turn (i.e. componentwise).

As you may suspect form the previous result, there is a simple general solution to obtain a consistent additive distance under any time reversible tree model of evolution. In its most compact form the ”General Time Reversible” (GTR) distance can be written as,

d = -trace(Pln[P-1F]) (*-*)

(Rodriguez et al. 1990). Here d is recovered as the expected number of substitutions per site (including multiple hits), and the ln function is the matrix logarithm of [P-1F]. The matrix ln function is the inverse of the matrix exp function, so ln(exp(Rt)) = Rt. So, given F we solve for Rt, and then take minus the trace of Rt. This distance was repeatedly derived by a series of authors (Barry and Harrington, 1987; Tavare, 1986)(Lanave et al. 1984, Rodriguez et al. 1990). Some of these authors where inaccurate or incomplete in their claims as to which models the distance would work under (e.g. Lanave et al. 1984), others used the distance but didn’t write it out as a single expression (Barry and Harrington, 1987; Tavare, 1986), while all but Lanave et al. appeared unaware of the previous work. For a fuller history of this distance see Waddell (1995), Waddell and Steel (1996) and Swofford et al. (1996).

While R has 16 entries, those on the diagonal are the just the sum of each row. Further, to have reversibility, the other 12 entries must meet 3 constraints (Tavare, 1986)(Waddell and Steel 1996) leaving 9 free parameters. Note, if you take (PRt). and multiply it by the sequence length (c) you get a matrix cPRt which is basically the F matrix corrected for multiple hits, and can best see where multiple hits are most frequent (Waddell and Steel 1996).

When dealing with finite samples (e.g. real data), P and F are replaced with their sample estimates (these are denoted by and) and of course distance d is then also an estimate (). Furthermore, is then replaced by F , a symmetrised form of (i.e. replace Fij and Fji with (Fij + Fji)/2 (Tavare, 1986). This is done to reduce sampling errors and F is shown to be an ML estimator of F under the model (see Waddell and Steel 1996). This result is also convenient, since a the matrix P-1F is guaranteed to be diagonalisable and will have real eigenvalues (Keilson 1979, see also Waddell and Steel 1996), so the matrix logarithm is easily calculated and defined as long as all the eigenvalues are greater than zero. Interestingly the nine free parameters of the time reversible model can be written in a very simple form so that R = SP , where S is a symmetric matrix (with 6 free parameters) and P will be the stationary base composition (with 3 free parameters)(Tavare, 1986). Zharkikh (1994) has chosen to call this an equal input related model, and alternatively R = P-1S, an equal output model. This distinction is confusing, and not useful, since any time reversible rate matrix R can be written as either SP or P-1S (where S is a different symmetric matrix in each case, see the proof in Waddell and Steel 1996). Neither of these representations are equal to R = S P-1 or R = PS, however these representations could be encountered amongst those who use the convention of writing the rate matrix R with columns, not rows, summing to one. Thus there is just one general time reversible model.

An interesting fact is that the general time reversible distance will also be exact under any model (time reversible or not) where the base frequencies are stationary at 0.25 each (Rodriguez et al. 1990, Waddell and Steel 1996). A similar result also allows for entries in R to change, as long as R changes in such a way as to keep the base composition stationary and the statement exp(S1Pt1)exp(S2Pt2) = exp((S1t1+ S2t2)P) remains true. This statement is generally only true if S1 and S2 have the same eigenvectors. A special case of this are any rate matrices for the Kimura 3ST model. These results gives hope that when base compositions are near stationary or especially near equal frequency, the distance (and rate matrix estimates) there will be some robustness to changing mutation pressures. An example of this is figure 3 of Waddell and Steel (1996), where the distances show up a varying ti / tv ratio in accord with ML estimates (Adachi and Hasegawa 1996) despite the assumptions of the model being violated in principle.

Modifying the GTR distance to unequal rates across sites

Golding (1983) showed that unequal rates across sites could distort distances, making them non-additive. This in turn leads to inaccurate edge lengths on trees, the potential to select the wrong tree if this factor is not taken into account (e.g. Jin and Nei 1990, Lockhart et al. 1996). While Golding (1983) and others have used a G distribution, invariant sites, for example, will also cause similar problems (Shoemaker and Fitch 1987), while Olsen (1987) considered a lognormal distribution. It has recently been proven that any distribution of rates across sites will lead to an underestimation of the true distance with time reversible models (Waddell and Steel 1996) and so potential inconsistency of tree selection.

Special cases of allowing a distribution of rates across sites when estimating a distance include allowing a G distribution of allelic substitution rates combined with a polymorphism distance (Nei et al. 1977), a G distribution with Jukes Cantor, Kimura 2P, and a 6 parameter TN distance (Golding 1983, Jin and Nei 1990, Tajima and Nei), a lognormal distribution of site rates with a Jukes-Cantor distance (Olsen 1987), and a mixtures of invariant sites and G distributions (Gu et al. 1995, Waddell 1995, Waddell et al. 1996, Waddell and Steel 1996).

The general solution to estimating time reversible distances when rates across sites are unequal is

d = -trace(PM-1[P-1F]) (*)

(Swofford et al., 1996)(Waddell 1995, pp 94, Waddell and Steel 1996), where function M-1 is known as the inverse of the moment generating function applied to a Matrix (Waddell 1995, Waddell and Steel 1996, and explained further below). In the case of the i.r. distance this reduces to the matrix logarithmic function (as described earlier); otherwise, make the same diagonalisation and apply the function M-1 to each eigenvalue in turn.

For the gamma (G) distribution, M[x] has the form [(a-x)/k]-a, and M-1[x] = a(1-x-1/a) (where a is the shape parameter of a G distribution with mean set to one, i.e. the probability density function, f[x] = a([xa]a-1e-xa)/G(a), where G is the gamma function, Steel et al. 1993, Waddell 1995, Swofford et al. 1996). Another interesting distribution is the inverse Gaussian, which has a shape like the log Normal, where M[x] = exp[z1-(1-(2x/z))0.5], and M-1[x]= 0.5z[1-1-(ln(x)/z)2](here z is the shape parameter, so a z this function goes to the logarithmic function, i.e. identical site rates, as it becomes small, e.g. 0.4, the distribution of site rates becomes increasingly skewed Normal). For rate matrices, the function M is always defined for any distribution with the mean set to one, however it may not be in closed form, in this case it is necessary to use numerical integrations to define M and M-1 (Waddell et al. 1996).

Another set of closed form corrections, M-1, are possible by allowing a portion of the sites to be invariant. For some sites invariant (pinv) and all other evolving at an identical rate, M[x] = pinv + (1 - pinv) exp[x], and M-1[x] = ln[(ri - pinv) / (1 - pinv)]. Modifying this so the variable sites are G (or inverse Gaussian) distributed gives

M[x] = (pinv + (1 - pinv) ((k - x) / k)-k) (or pinv + (1 - pinv) exp[z1-(1-(2x/ c))0.5]),

and M-1[x] = k (1 - [(x - pinv) / (1 - pinv)]-1/k) (or 0.5c[1-1-(ln[(x-pinv)/(1- pinv)]/c)2]),

for these two distributions respectively (Waddell 1995, Waddell and Steel 1995, Waddell et al. 1996). Further, if the base composition of the invariant sites is distinct from that of the variable sites, the proportions of invariant sites can reflect this, by rescaling of F, before applying any further transformation, as described further below for the LogDet (Swofford et al., 1996)(Waddell 1995). The basic operation is replacement of F by (F - pinvPinv) where pinv is the proportion of sites assumed to be invariant and Pinv is a diagonal matrix of the estimated base frequencies of the invariant sites. The distribution of the site rates, lj, could in reality take just about any form. However, the normalization that the mean rate is fixed to one (so we get expect number of substitutions per site and not some multiple of this number), means that if site rates are to be highly unequal, the most distributions will increasingly have an approximate L shape (see Waddell 1995, Waddell et al. 1996). An interesting feature of the invariant sites distributions is that they allow a distinctly bimodal character to the rates across sites, and in some cases this results in a better fit that the continuous distributions (e.g. the G, see Waddell et al. 1996, Waddell and Penny 1996). An important feature of the continuous distributions is that many have a tail that suggests increasingly rapid substitution rates in the 'tail'. Alternatively, removing invariant sites produces a very strongly non-linear transform as pinv approaches the observed proportion which are unvaried (see figure 1 above, Waddell et al. 1996). This can result in some quite distinct differences between various distributions (e.g. see Waddell and Steel 1996 for examples of the effect on ti / tv ratio and divergence time estimation). Thus, choosing a distribution of rates across sites should be treated with as much consideration as any other aspect of the distance model.

To understand what function M-1 is, it is best to understand what M is and how it is involved in estimating F. (that is we will start with the model and predict the divergence of two sequences]. Earlier we stated that F = P(exp [Rt]) for the i.r. model. Let's now assume that the rates of evolution of sites fall into two classes, with proportion 1 (p1) fast (rate l2) and proportion (p2) slow (l1) (and the mean rate is fixed to 1, so p1l1 + p2l2 = 1). In this case F(overall) is a mixture of two F matrices, p1Fl1 + p2Fl2 = p1P(exp[Rl1t]) + p2P(exp[Rl2t]) = P(p1 exp[Rl1t] + p2 exp[Rl2t])

= P(E[eRljt]). This last function (in brackets), is the expected value of eRljt , known as the moment generating function in statistics. Since exp[R] can always be calculated via diagonalisation, this gives

F = PM[Rt] (*)

for any time reversible model, or for any rate matrix along a single directed edge (Waddell and Steel 1996). So M[X] simplifies to applying function M[x] = E[eljx] to the eigenvalues of X, in place of the more common function exp[x] used when sites have identical rates. Rearranging these equations gives equation * above, and M-1 is just the left functional inverse of M (i.e. the function M reflected in the line x = y, just as the logarithm is the left functional inverse of the exponent).

Rate matrices can also be estimated under the general time reversible model for pairs of sequences, as in the i.r. case. When allowing for unequal rates across sites, the general tendency is to infer more multiple hits amongst those entries in the rate matrix which are already largest. If the entries in R are of very different sizes (as they typically are for mtDNA) this implies a lot of extra extrapolation based primarily on a few types of change which may already approach saturation. As a consequence unequal rates across sites can see a large increase in the variance of estimated distances and hence potentially a marked decrease in bootstrap support for at least some aspects of the model (either the tree topology or edge length estimates). This is not a disadvantage of the method per see, but simply a recognition that the sequences do not contain the resolving power that one might otherwise assume they do.

16.18 Families of interrelated distances

There are many possible submodels of the general time reversible distance and since all of these can be written in the form of R = SP, where S is a symmetric rate matrix, while P is as defined earlier a diagonal matrix with diagonal entries, x, in the range 0 < x > 1, and summing to one. Here, whatever P is set to will define the stationary base composition for the Markov process with rate matrix R. Thus, constraints on the GTR model can be made in either S and / or P (Table 1). This framework describes most of the well known closed-form distance estimators. It includes all the distances shown in the diagrams of Zharkikh (1994) and Swofford et al. (1996) and more.

Table 1. Time reversible distances and their inter-relationships

Note: * indicates important work describing the model the distance estimate is based on, but not deriving a closed form distance estimator, while + indicates not exactly the same model.

Form of S Most gen. assmpt.in S From of p Described by: Site to site rate variability Described by: Jukes-Cantor (1969) - a a a a - a a a a - a a a a - all substitution probabilities equal equi-frequency

AT GC

A C G T Jukes and Cantor (1969) —-

Felsenstein (1981)* Tajima and Nei (1984) TaverÈ (1986) gamma p.d.f.

inv. Gass.p.d.f.

invariant sites model Golding (1984)* Jin and Nei (1990) —-

—- Kimura 2P (1980) - a b b a - b b b b - a b b a - sag, sct sac, sat, scg, sgt equi-frequency

AT GC

A C G T Kimura (1980)

Tamura (1992)

Felsenstein (1984)* Hasegawa et al. (1985)* Tateno et al. (1994)+ gamma p.d.f.

inv. Gass.p.d.f.

invariant sites model Golding (1984)* Jin and Nei (1990) —–

Shoemaker and Fitch (1986)* Kimura 3P (1981) - a b g a - g b b g - a g b a - sag, sct sac, sat scg, sgt equi-frequency

AT GC

A C G T Kimura (1981)

—-

—- gamma p.d.f.

inv. Gass.p.d.f.

invariant sites model Waddlell(1995)

″ ″

″ ″ Tamura and Nei (1993)

- a1 b b a1 - b b b b - a2 b b a2 - sag sct sac, sat, scg, sgt equi-frequency

AT GC

A C G T —–

—-

Tamura and Nei (1993) gamma p.d.f.

inv. Gass.p.d.f.

invariant sites model Tamura and Nei (1993)

—-

—- Lanave et al. (1984)

- a b g a - d e b g - f d e f - sag sct sac sat scg sgt equi-frequency

AT GC

A C G T —-

—-

Lanave et al. (1984) Tavere (1986) Barry and Harrington (1987) Rodriguez et al. (1990) gamma p.d.f.

inv. Gass.p.d.f.

invariant sites model Waddell (1995) Waddell and Steel (1996)

″ ″

Studying this table, some interesting trends emerge. The first column shows the basic form of the underlying S matrix. Initially there was a trend through time to make it more general (indicated are the authors who first derived a distance incorporating a more general assumption regarding the form of P). Occasionally, however, there has been a reversion to a simpler model e.g. Tamura and Nei (1993) identified a circumstance (e.g. mtDNA) where it might make more sense to group a lot of the rarer changes (the transversions), but differentiate two classes of transitional changes.

The second column of the table looks at possible constraints on P. In reality, of course, each base is likely to have a different value, but in various circumstances some of these can grouped in order to reduce the complexity of the model. This grouping can also allow for the derivation of distances, in a simple closed form, e.g. Tamura's (1992) distance forces the assumption of equal contents of A and T, and equal C and G, but a closed form does not exist if S is of the form of Kimura 1980, and base frequencies are unequal (the HKY model, though Tateno et al. derive a distance for the related F84 model described later).

The last two columns are to do with site to site rate variability, as described earlier. described in more detail later. Suffice to say all these distance estimates can be modified in closed form to allow certain types of site to site rate variability (particularly, a gamma distribution, or an inverse Gaussian distribution, or either of these mixed with invariant sites). As mentioned earlier and discussed later, the invariant sites may not reflect the base composition of the variable sites.

There are also many other ways of grouping changes in addition to those shown in table 1. This number is the number of distinct ways of choosing x (x = 1 to 6) from 6 (the entries in S) = 203 (a type 2 Sterling number), multiplied by y (y = 1 to 4) from 4 (the entries in p) = 16, so x x y = 203 x16 = 3248 possible distinct evolutionary models! Clearly too many to write down, but each potentially as valid as any shown in table 1, and each giving a different distance estimate. Many of these distances do not have a closed form solution (e.g. Zharkikh 1994, Yang 1994), but an approximate and an iterative distance is available for all of them and is described later. Closed form solutions are absent when the eigenvalues of the expected F matrix cannot be written down in closed form. Since our purpose here is not to talk about specific distances, but rather give biologists an overview of what is possible and appropriate in an analysis, we will not write out all the known closed form distance estimators, rather for compilations of these see Gojobori et al. (1990), Zharkikh (1994) and Swofford et al. (1996).

There are thus many possible time reversible distances, even with only 4 states. This also means the biologist must consider the patterns in their own data and accordingly make decisions on what a reasonable approximation to S and P will be.

16.19 Restricting the GTR distance by weighting in matrix F

It is sometimes possible to constrain a more general to a more specific model by forcing the data into the configuration expected under that special case. For example, whenever the Kimura 2ST model is in operation, the likelihood of pairwise patterns like AG, GA, CT and TC to be equal (since all transitions are equally likely) while all the transversion patterns AC, AT, CA, CG, GC, GT, TA and TG are also equally likely. Grouping (or averaging) directly patterns with equal likelihoods forces the data to what is called a sufficient statistic (a form of data that contains all the information of the original sample under the model).

Based on this statistic, there is a unique transformation of the data leading to an ML estimate of the distance. Such transformations leading to ML distance estimators are limited to a few special cases with 4-states, these being the Jukes Cantor distance, the Kimura 2ST distance and the Kimura 3ST distance (e.g. see Zharkikh 1994). There are also two other closed form ML distance estimators with 4 states. One is the LogDet - Paralinear distance (Lake 1994, Lockhart et al. 1994) based on the sufficient statistic of any i.r. and i.i.d. model (i.e. matrix F), while the second one is the general time reversible distance based on symetricised F (Waddell 1995, Waddell 1996).

When it is not possible to group the data so as to give rise to a single sufficient statistic, the data may still be grouped based on an expectation which does not capture all of the information in the original F, but which does force it to meet the models expectations. Lewis and Swofford (1996) show how to generate such restrictions of F. This gives F under the constrained model exactly in certain cases (apparently corresponding to any F matrix which has closed form eigenvalues, or an approximation to F in all other instances). Following this constraint, application of the GTR distance transformation to the special F gives a distance estimate under this model.

The F matrix of any time reversible model can be written as F = PP, and P may have the form of XP (here X is a symmetric matrix with off-diagonal elements in the same arrangement as the corresponding S rate matrix). Thus F = PXP. Such an F matrix is always symmetric, in accordance with the expectations of a time reversible process. When dealing with (i.e. from sample data), first symmetrize to give F , then force the model into the form F = PSP. So if we go to table 1, and we want to force F to meet the expectations of the Tamura-Nei model, look up S for this model, and average the elements in F which are equal in S, we will call this matrix F+. Next, it is necessary to take into account the pre and post multiplication by P. Using P from F , force P to meet model expectations (again by unweighted averaging), to give Pw, then pre and post multiply Fw, to give PwFwPw. The last step is to renormalize the previous product so that all entries sum to 1. This is done by dividing all diagonal elements by pi2 and all off diagonal elements by Spipj/ n, where the sum is over pipj for all elements in Fw which have been averaged, and n is the number of cells averaged. To put it cell by cell, F*ij = p+ip+j (Fij + Fji + SFwmn)/(pwipwj + pwjpwi + Spwmpwn), where index m and n are for all cells in S which are expected to be equal. For example, under the Tamura 1992 model, to estimate FwCT, first generate pw by averaging pa and pt, plus pc and pg, then, . Lewis and Swofford (1996) call this a method of moments estimate of the entries in F, so after applying the distance transformation (Eq. *), it gives rise to a method of moments distance estimate. As noted above, in only a few cases is this distance a pairwise ML estimate.

This same principle is used in other circumstances in phylogenetics. One example is to force the 4-state Hadamard conjugation (Hendy et al. 1994) which naturally operates on a generalised form of the Kimura 3ST model, to make corrections under a Kimura 2ST or Jukes-Cantor model just by averaging those site patterns which are equally likely under these more restrictive models (Swofford et al., 1996)(Waddell 1995, Hendy and Penny 1996). Hadamard conjugations can be though of as a distinct case of distance corrections. Another example is in the use of LogDet distances, discussed below.

16.20 Estimating distances for time reversible rate matrices not in the form SP

Some interesting time reversible distances have rate matrices which do not appear above in table 1, and at first glance may not appear definable as SP. However, the proof in Waddell and Steel (1996) shows they must be able to written this way if they define a time reversible model. Two notable examples of such rate matrices are the four parameter model of Takahata and Kimura (1981) and the five parameter model of Felsenstein (1984). Considering the latter first, it has rate matrix as given in Eq. * of chapter * (on ML), so we can separate S and p as:

S = , p = [pA, pC, pG, pT]

It is possible to then derive a weighted F+ matrix corresponding to this model.

For the 4 parameter model proposed by Takahata and Kimura, the situation is slightly more complicated, R =

However, R of this type gives rise to p = [g/(2(a + g)), a/(2(a + g)), a/(2(a + g)), g/(2(a + g))] (Zharkikh 1994), and once p is defined it becomes clear that S must be

S =

(Waddell and Steel unpublished) so the multiplication of SP gives R. Given this, it is now possible to apply the weighting described by Lewis and Swofford (1996). This last model is useful, since it may best capture major aspects of pseudogene rates of substitution with just four parameters (Takahata and Kimura 1981).

Another way to view this last model, is in p equate pa = pt and pg = pc, (so their ratio is g / a), while in S fix sac = sgt, sag = sct, so the relative size of sat to scg is equal to a / g = pc / pa. Thus, in both these models, parameters in p are feeding back to give an additional constraint in S which is not defined directly as entry x = entry y (or put another way, parameters in S and P need not be independent). Here is an example of a three parameter time reversible model, where all parameters appear in both S and P. S = , p = [pa, pc, pg, pt]

(and pT = 1- pA - pC - pG). Here R = SP has all 16 entries different, but only 3 free parameters governing them all. Indeed, if the entries in S can be written as any function of the p's, there are an infinite number of such models possible. However, some like that of Takahata and Kimura (1981) may be particularly apt for describing the substitution process in some genes. Picking useful rate matrices of this type, will be a problem, but their use for distance calculations (or likelihood) can be semi-automated in a program like PAUP*. This issue of identifying a good model distance with few parameters is considered later.

A catch is that the averaging technique used above only applies exactly in some instances, specifically, when the form of the transition matrix P can be written explicitly under the model. If not, it can give an approximate distance which converges to being exact as d tends to zero. As d becomes larger, caution is necessary in interpreting the distances obtained as either model exact or additive under any model. Fortunately, iterated ML estimators can fill in these instances, but require estimates of parameters in R. Lastly, being able to derive many time reversible distances by constraints on F, leads to an integrated approach for calculating the variance of each, as described later.

16.21 Iterated ML distances

Some of the above distances are ML estimators but only for a pair of sequences at a time (and when R = SP is fully optimised, Waddell 1996). If we know more about R we can implement more specific ML estimators. There are presently two main cases of iterated ML implemented in PAUP*:

(1) Iterated ML fixing the precise entries in R for all distance estimates. Given good estimates of entries in R is expected to yield the lowest sampling errors, and is preferred method when model has homogeneous R and is time reversible.

(2) Iterated ML specifying just the restrictions on R, but estimating the values separately for each pairwise comparison. Will have a higher variance than the former approach, but a lower variance than otherwise equivalent non-ML distance estimators (e.g. TN-94 distance). Convenient in that no numerical values in R need to be specified, and should have some added robustness when model is non-homogeneous, and perhaps when non-stationary.

Either method is often implemented by specifying or optimising entries in S, and taking P to be close estimates already (this is done in Felsenstein 1993, by asking the user for a ti / tv ratio). While it has yet to be shown with simulations, from theory, we expect that pairwise ML distances (especially when fixing R for all distances) should generally perform at least as well as the ad hoc weighting schemes proposed for some distances (e.g. Schrˆniger and von Haeseler 1993).

While iterated ML is computationally more complex that the time reversible distances previously mentioned, it will give smaller sampling variance than the method of moments approach. The type (1) distances do require a reasonable estimate of R (or just the S part). Estimation of the free entries in the S matrix can be made efficiently by ML on sequences (evaluation on a near optimal tree should usually give reasonable estimates). Suppose that R appeared close to the form of the Felsenstein (1984) model. It seems reasonable to assume that the observed base frequencies of P were adequate estimates of their true values. Doing this, leaves only the form of S unknown. In the F84 model, the only remaining free parameter in S is k or kappa.

Felsenstein (1993) introduced iterated ML estimates for the F84 and K2P models. It is possible to use the same general procedure for any GTR model. F for any time reversible model, can be predicted using the equation

F = PM[Rt],

(Waddell 1995, Waddell and Steel 1996), where M reduces to the matrix logarithmic function when site rates are identical i.e. F = Pln[Rt]. So, enforcing the assumptions about P and S, predict F, and measure the likelihood of the data as cSFijobs(ln[Fij])(the standard log likelihood function, where c is the sequence length). We then iterate t and any free parameters in S and P until the likelihood is maximised. For those more familiar with standard biometry, the minimisation of the likelihood is equivalent to the minimisation of the likelihood ratio statistic G2 (called G by Sokal and Rohlf 1981). We could equally aim to minimise the X2 statistic S(Fobsij - Fij)2 / Fij (e.g. Blaisdell 1983); this statistic having very similar properties to likelihood with multinomial data. Once the minimum is found, the estimated evolutionary distance is S(PiRijt) i j = -trace(PRt).

As long as our estimates of the parameters being fixed are "good", this iterated ML method is expected to yield distances with reduced sampling variance (and often reduced bias). Perhaps the best way to get estimates of parameters in R, given more than a pair of sequences, is via ML on sequences (Felsenstein 1981, Ritland and Clegg 1987, Yang 1994). This may seem ironic, in the sense that one could ask why not do ML directly. One reason is if you have 60 sequences, you may not have the ability to evaluate ML on a single tree of all these taxa, but analysing just 20 (with a reasonable estimate of the tree) could provide a good estimate of entries in R and the starting point of an extensive distance based tree search. The estimates on parameters in R do not need to be pinpoint accurate to realize gains in reduced variance, and should be a lot better than ML one pair of sequences at a time. Thus alternative ways of estimating entries in R, or using ML to estimate R on a subsets of taxa, could all be useful in different contexts (e.g. the number of taxa).

An exploratory way of using these distance estimators, is to first bootstrap trees using the GTR distance, then repeat the analysis enforcing specific restrictions to the model using the method of moments, then the iterated ML estimators. With assumptions about R that seem realistic (see below) this can give a qualitative feeling for how much advantage the iterated ML distances may offer (if the restrictions are justified). To make a valid comparison of bootstrap proportions with different methods, you should run paired tests (i.e. exactly the same data, the same random number seed etc.) preferably of 1000 or more (the exact number required for a certain accuracy requires knowledge of how often the methods disagree on a specific branching order).

Computational speed is not generally a concern with any of these distance estimators (even when bootstrapping); the process of tree searching takes the great bulk of the time in most instances. When using more and more complex models, the degree of non-linearity increases. Non-linearity in the transformation of distances results in increased sampling variance, beyond a simple scaling factor and also a bias in the distance (towards being too big, given that no samples are rejected, Waddell et al. 1994). This effect carries over to bootstrapping, where it will act to increase the conservativeness of the bootstrap for estimating edge length support. Thus the reduced sampling error and bias of iterated ML distances when fixing R, should relieve this concern.

Estimating R and deciding when to group entries in either R or F

Importantly, the sampling variance of time reversible distances estimated using the iterative ML approach (and with also with certain, but not all, weightings of F) can be decreased by homogenizing rates in (and so reducing the number of free parameters). For example, if entries in either or are not significantly different, then it makes sense to average these values. Each time this is done it will reduce the number of parameters by 1. However, there are many choices to be made in doing this, and we now consider the relevant issues, then finally make some recommendations.

Before we delve into this critical question, it is worth noting the different requirements of iterated ML distances and direct transformation distances. While both require a decision on which entries to group, but the former, then also requires an estimate of the relative sizes of all entries in R (which, assuming reversibility, can be considered as SP). Here, we will consider this problem in three steps: (1) Consider a simple example, where an estimate of R is provided by the equation R = M-1(P-1F) and which entries can be grouped with minimal effect on additivity. (2) Consider tests to determine if a particular reversible model is suitable for the data. This is a very difficult question, both statistically and logically. We could aim to measure additivity, or we could just ask do the predictions of a particular distance model hold (both approaches are considered) (3) Consider good ways of estimating R for the iterated distances which require them.

(1) A simple example of grouping entries in R.

Considering, t from figure 2D of Waddell and Steel (1996), we next infer the form of = , which gives = (note that here p is the equilibrium vector of , such that pt = [0, 0, 0, 0], which gives p = [0.28, 0.35, 0.11, 0.27] in this instance). In this example, which proceeds after the removal of apparently invariant sites, it becomes clear that pA and pC are more unequal than in the average over all sites, but pA and pT are now more similar and one might consider averaging them. Within the matrix, no entries are exactly equal, although 0.0005, 0.0085 and 0.0151 come closest. Homogenizing these entries, gives a new, previously undescribed distance in the time reversible family.

In choosing which entries to homogenize in the previous example, it may seem that proportionally, transition entries 1.4453 and 0.9212 are much more similar than any pair of transversions. However, the much increased frequencies of these substitutions in this example will result in a more substantial underestimate in the overall distance if homogenized. Thus even without seeing the covariance matrix of it makes sense, as a rule of thumb, to homogenize entries in order of their absolute size difference (starting with the smallest difference). Interestingly, if we estimate for the comparison of orangutan to siamang sequences, the same three transversion entries mentioned earlier are still the most similar in size, although now they differ by up to 0.1. They would still be the first entries to homogenize, but the difference in distance after homogenization would be more noticeable, making such homogenization of arguable value. So the salient point here is that homogenizing the smallest entries in R or F will generally have the least potential affect on additivity.

16.22 (2) Testing which reversible model fits best.

Additivity: When homogenizing entries, it would be nice to have a guide to how much the additivity had changed. This is a surrogate answer to the critical question "does the simplified distance improve the ability of tree estimation?" This second question cannot be answered authoritatively, as explained later. Perhaps the ideal solution to measuring additivity of distances and simultaneously estimating R would be using an ML method based on just the pairwise distances. As discussed earlier this rapidly converges (with increasing sequence length) to using iterated GLS to measure degrees of fit of data to model. For the GTR distance, one could fix R for the whole tree, then estimate their values so as to minimise the overall GLS SS statistic. This should be c2 distributed under the model, and if it is too large, the a reversible distance model may be inappropriate. Then, starting with a more general model the effect of the imposition of constraints such as pa = pt, or sat = sag is gauged to be statistically undetectable to additivity if the residual c2 statistic increases by less than 2.0 (this is basically the same approach used by the Akaike criterion, e.g. see Kishino and Hasegawa 1990).

Since iterated ML on distances is not available, ML on sequences might be considered (e.g. measure the fit of data to model, after reoptimising a constrained form of R). However, the apparently increased sensitivity of ML on sequences verses ML on distances (Waddell, 1995) may result in a problem: the sequence based method is expected to be biased towards saying the simplification of the model is causing a problem before it causes a clear decrease in distance additivity. ML on sequences is also prone to distortions of the distribution of the likelihood ratio statistic (G2) due to sparseness and tree selection (McCullagh and Nelder 1986, Reeves 1992, Goldman 1993) so we must hope that expecting an increase 2 units for each parameter which is removed, remains correct (e.g. McCullagh and Nelder 1986). Further, since we do not even know which two models we wish to compare, we face a very vexing question of model selection. Trying many alternative models can dramatically distort the expect c2 distribution of the fit statistic for a single run, and we would run the risk of simplifying the model too much (or if selecting a more complex model starting with a simple model, choosing to complex a model, Miller 1990, Waddell 1995). The point here is just to be aware of these problems, while later we make some recommendations.

The situation becomes more complicated if we are going to impose constraints on F, rather than directly on R as with iterated ML distances.

The problem is manifold. Firstly, the distance estimators may not be statistically efficient, so the c2 type of test may be distorted. Secondly, we would need to take into account the true number of parameters in the model. For example, if you take the restriction of going from Kimura 3P to 2P distances, you are potentially making not one constraint to the tree model, but up to one constraint per edge in the tree (= 2t-3 constraints) since these distance estimators are ML estimates even when these rates fluctuate on each edge of the tree. Thirdly, if the model is not equal frequency (with the root base composition in equilibrium), then R should not vary from edge to edge or distance to distance, but it clearly will if we make individual estimates (e.g. see figure 3 of Waddell and Steel 1996), so the effect of constraints would be unpredictable on the fit statistic.

Similar tests could also be carried out with uniterated GLS (e.g. Bulmer 1991, Rzhetsky and Nei 1995), or even with WLS. All these doubts about the appropriateness of such tests will impinge upon these tests also, not to mention the requirement that they be programmed and available. Hopefully, however, these tests will be shown to be useful.

Direct tests on F and P. These simpler and less direct tests can also help to guide us in the choice of a reversible model. Firstly, a test can be made of which of the base compositions (entries in P) are statistically indistinguishable, so allowing a selection of a simpler distance. You might think this is easy, but it actually not straight forward for more than a single sequence. For a single sequence, the test is fine, after assuming an i.i.d. model, count the frequency of the four bases (the observed data) and represent them in 4 contingency table cells. Next, average those entries which you suspect are statistically equal, place them in adjoining cells and perform a X2 test (with one degree of freedom for each pair of cells averaged). However, if the base composition is being measured on two sequences, then a problem arises since the sites will be correlated due to their history, if they are clearly related. For two or more sequences, this correlation of changes can be solved by using GLS, but if not taken into account.

Rather than testing P and S separately, a better approach appears to be to test F against expectations. It is logical to first test the expectation that F is symmetric, and so consider whether the data are showing a distinctly non-time reversible character (TavarÈ 1986, Waddell and Steel 1996). Using either a X2 or G2 test statistic is reasonable (e.g. Read and Cressie 1988). The X2 test statistic is , while the G2 test uses 2. Both test statistics asymptotically have a c2 distribution with degrees of freedom (d.f.) equal to number of entries i j (12) minus the number of estimates made in F (6), which leaves 6 d.f. However, a potential problem with this test is its lack of power when a molecular clock is likely, since this too implies that F is symmetric (e.g. Waddell and Steel 1996). This type of test concentrates on just the sites that show change, and so for a single pair of sequences, it is unaffected by invariant sites or correlations. However, for testing more than 2 sequences, correlation structure is again a concern.

Table 2. The G2 test statistic for the symmetry of F matrices (6 d.f. each, so 11.05 has P = 0.05). Numbers in bold indicate P < 0.01. Those in italic showed substantial decreases in size after appropriate grouping of entries.

H C P G O H C 9.67 P 11.61 9.67 G 8.64 6.32 5.43 O 9.30 19.88 21.89 17.92 S 3.58 6.63 5.50 3.10 13.71

To deal with correlation structure a GLS type of test can be invoked (e.g. Rzhetsky and Nei 1995). However, for simplicity and robustness we suggest the following test. The aim is to make pairwise tests of the symmetry of F, being careful that the paths through the tree connecting the sequences do not cross (i.e. share no edges in common), so they are uncorrelated. Start with a reasonable estimate of the tree to be tested. Sometimes, as in the case of the mtDNA sequences of apes (Horai et al. 1992, Horai et al. 1995), it is nearly certain the biological tree is ((((Human, (Chimp, Pygmy chimp)), Gorilla), Orangutan), Siamang) (from both tree reconstruction on these sequences and prior knowledge). At other times it is less certain but often at least substantial parts of the tree are biologically ″known″. Further, if only short internal edges are crossed more than once (e.g. edges with low bootstrap support) then the impact upon the test is expected to be minor (as correlations will be small).

With the example data, the 3 nonintersecting paths through the tree are (Chimp-Pygmy chimp), (Human-Gorilla) and (Orangutan-Siamang) (see G2 statistics in table *). If the test is done with the G2 likelihood ratio statistic, then by summing up the results (and degrees of freedom) for each pair an overall test can be made (e.g. Sokal and Rohlf 1981, in this case it is 9.67 + 8.64 + 13.71 = 32.01, 18 d.f. P = 0.022, or 4.66 + 8.64 + 13.71 = 27.30, 16 d.f. P = 0.038 with grouping). This test is robust and practically unbiased as long as no expected cell count is less than 1, and 80% of cells have an expected count of 5 or more (if necessary, group cells to achieve this). There will be a slight drop in power from the ideal test, a function of the sum of the edges not crossed in the paths. This test is insensitive to sites having unequal rates, or sites having distinct substitution processes and / or changing relative rates on different edges (as long as all sites evolve by any reversible model). Thus, this test gives clear evidence that this data is not evolving by a reversible model, a result which will hold, even if ML suggests sites have changing relative rates (e.g. Adachi and Hasegawa 1995). The result corroborates the claim of Adachi and Hasegawa (1996) who suggested deviations from the model due to orangutan and possibly human. Additionally, checking the table of results, we see that it is orangutan which appears most anomalous. To check the edges not yet included in the test, and to see if exclusion of the orangutan brings a non-significant result, other non-interesting sets of paths can be tested (e.g. from siamang to gorilla plus human to chimp, in this case giving 12.77, 12 d.f. P = 0.386, or 10.00, 11 d.f. P = 0.530 with grouping). So, for these taxa (and edges in the tree), there is no suggestion of the model being non-reversible. It is also possible to directly test a related aspect of reversible models, that is stationarity of base composition (or marginal homogeneity of the F matrix, which is a type of contingency table, e.g. Ireland et al. 1969). For sequences x and y, this is testing that p(x) and p(y) are equal. A generalised least squares test requires the variance covariance matrix (V) for the differences of base compositions. This has variances equal to (p(x)i + p(y)i - 2Fii) / c2, and covariances - (Fij + Fji) / c2. The difference in base frequencies is d = p(x) - p(y), and the GLS SS deviation is, SS = dtV-1d (which asymptotically, as c , has a chi-square distribution with d.f. = no. elements in p -1). This is sometimes called Stewart's statistic (e.g. Ireland et al. Eq. 1.4). For marginal entries, minimum G2 fit statistics are also possible, but do not have a closed form solution. Applying this test to the hominoid data of Horai et al. (1992) data, gives significant results for orangutan deviating from base composition equality with the other species (Waddell 1996), and this tips the overall results for non-intersecting paths through the tree in favour of rejecting base homogeneity.

GLS SS or Stuart Statistic d.f. = 3 5% = 7.81, 1% = 11.34 H C P G O H C 2.52 P 4.94 4.92 G 3.80 0.28 3.25 O 4.01 10.73 9.09 12.59

David I can cut this table, just thought you might like to see the numbers.

Some (e.g. Kumar et al. 1993) have tested base composition homogeneity with a standard X2 test of cp(x) = cp(y). Even for just two species, this test behaves erratically, and can be either far to conservative, or far to liberal, depending on the form of F (Waddell 1996). A further advantage of the GLS SS test used here is that it is not affected by the presence of invariant sites.

If the tests fail to reject a time reversible model, it is then logical to use similar tests to compare the expectations of different restricted reversible model. For all time reversible models, F = PP, and in some instances, P is known in closed form (see above). For example, constraining F* to met the expectations of the Tamura-Nei distance, then testing it for fit to the observed F, gives a G2 statistic of for the paths CP, HG and, OS ($P < 0.0001$, and unlike the GTR model, the test still rejects this model when the orangutan sequence is deleted, $P < 0.0001$). If F* is not available in closed form, then for any reversible distance, F* is predicted as PM(Rt) and G2 is minimised when iterated ML distances are used. Choosing a model is precarious, in that the more comparisons are made, the more the problems of multiple tests should be considered (e.g. see Miller 1990).

Estimating R for use with iterated ML distances

A good way of estimating entries in R is by via iterated Generalised least squares. Another, is via ML on sequences (e.g. Felsenstein 1981, Ritland and Clegg 1987, Yang 1994). In contrast to a previous example, ML is not being used to test whether a certain model will give significantly closer to additive distances, just an estimate of R which is hopefully close to accurate. A less efficient way of estimating the entries in S for any submodel is to simply average the values obtained by GTR ML. While these values will not be optimal, in many instances they should be close enough to cause minimal distortion to additivity.

It is also possible to estimate R as a sum of the R matrices from all pairwise comparisons using Rt = ln(PF*) (where F* is F constrained to meet a certain models expectations, see section * earlier). Here weighting of averages is desirable but requires estimates of the variance and covariance matrix of each R (Waddell and Steel unpublished manuscript). A third way, based on parsimony reconstructions (Yang and Kumar 1996) also seems promising. It seems likely that estimating entries in R will not be a serious problem for distances. Further, any fluctuations in the smaller elements in R are expected to have only small impacts upon the inferred distance.

For the last word on applying constraints to R (either iterated or by direct transformation of F), a set of simple guides are useful in exploring the implications for your data. The least impact on additivity is expected when you start grouping entries in S (and P) which have the smallest absolute difference in size. For example, if sat and sgt are 0.01 and 0.02 respectively, then their grouping will have much less impact on additivity then averaging sag and sct if their values are 0.8 and 0.95, despite their being proportionately much more similar. Progressing in this way from a general to a specific reversible distance (perhaps all the way to the Jukes-Cantor), it may be useful to note if and how the tree changes, and also how the bootstrap support for nodes of interest change with each subsequent homogenization. A very general i.i.d. transformation, the LogDet or paralinear distance

If your data clearly violates either symmetry of F and / or unequal base composition, it is desirable to consider a distance which applies to models which can show such non-reversible characteristics. At present the best known, simplest, and perhaps most robust such distance is the LogDet or Paralinear distance (Lockhart et al. 1994, Lake 1994). The hope is that by using such a distance, robustness will particularly be gained to the phenomena known as ”edges attract due to unequal base composition” (Lockhart et al. 1994), or the equally problematic ”long edges repel due to base composition” (Waddell, 1995). The first factor is prevalent with observed distances and parsimony. If two unrelated sequences in a tree both independently begin to enrich in AT for example, then there will be many multiple hits resulting in parallel of convergent patterns tending to group these species, and they are artificially drawn together on the tree. Additionally, the longer the two AT rich edges are, and the more AT rich they become, the stronger this effect.

Conversely, ”long edges repel” comes into play when two sequences, perhaps sister taxa, undergo opposite trends in their base composition (e.g. one becomes AT rich, the other AT poor). A profound affect of this is that the proportion of sites at which these two species differ can rise much more rapidly and even become larger than expected under a stationary model. Application of a power transformation to the F* matrix of these species (e.g. any reversible distance) can suddenly see the distance between them vastly overestimated (either in absolute terms, or in relation to the distance to other taxa). Ironically, in some such cases, the observed distance can be more accurate than any time reversible distance both in terms of estimating the true distance and in terms of best additivity on the correct tree (Waddell, 1995). An example of this effect is detailed later.

The LogDet is a transformation that yields additive distances under a much wider set of models than any of the standard distance transformations (which typically assume a homogeneous first order Markov process, with stationary base frequencies). It will yield an additive distance under any identical rates (i.r.) and i.i.d. Markov model of evolution. This general model is described by a tree, with a root any where on it, and this root can have any base composition (as long as all states have a non-zero probability of occurrence). Each edge on the tree has a probability transition matrix (P) that contains the substitution probabilities for all sites. The only requirement of this matrix is that it is not "more evolved than random"(which implies it has a positive determinant; it need not correspond to any continuous time process so is considerably more general than a different 12 parameter R matrix on each edge). The P matrix on each edge of the tree can be completely unlike the P matrix on any other edge of the tree. Each P matrix will have associated with it its own set of stationary base composition values; that is the base composition that a sequence evolving by Pi will head towards if the same Pi is applied over and over (i.e. after many multiple hits per site, although most of the convergence towards a new stationary base composition tends to occur by the time each site has expected one substitution). In practice, perhaps most importantly, the LogDet is able to return additive distances under a model where base composition varies across a set of sequences, something which in general will throw all the standard distance corrections into giving non-additive distance estimates, which can then mislead tree building algorithms into selecting an incorrect tree.

The basic LogDet transform is

dmn = -ln[det(Fmn)] (*)

where square matrix Fmn contains the proportion of sites in sequences m and n showing the different possible pairings of states, i.e. the ij-th entry of Fmn is the proportion of times sequence m has state i aligned next to state j in sequence n. (e.g. if i = A aligns next to j = T 11 times, and the whole sequence is 100 sites long then this entry in F will be 11/100 = 0.11). With such Markov processes, the det or determinant of a matrix can be thought of as a one dimensional measure of how far towards random a matrix is (a geometric interpretation of the determinant is the volume of space that an n n matrix encloses in Euclidean space). For example the determinant of a 2 2 matrix (such as counting just transversional changes would yield), is just (1-a) (1-b) - ab. So the two state LogDet distance is just, dmn = -ln((1-a) (1-b) - ab).

Work by Steel (1994) proves that this measure is additive under any i.r. and i.i.d. model, but why should this be? Recalling that Fmn = PtrmPrPrn, then det(Fmn) = det(PtrmPlcaPrn) =

det(Ptrm)det(Plca)det(Prm), where lca is the last common ancestor of sequences m and n (this rearrangement is possible because the order of multiplication of determinants is distributive, commutative and associative). So the term -ln[det(Fmn)] can be rewritten as -ln[det(Plca)] + -ln[det(PtrmPrn)]. Further the last term can be written as -ln(det(Ptm1)) + -ln(det(Pt12)) + ...+ Ptk,lca + Plca(k+1) +.... + -ln(det(Pln)) (where k is the label of the last internal node on the path from m the root and l is the last internal node on the way from r to n) (Barry and Harrington, 1987)(see Cavender and Felsenstein 1987 Steel 1994, and Lake 1994 for a mathematical description of this property applied to tree based sequence evolutionary models). The taking of logarithms has converted a multiplicative property of a estimate of the amount of change on each edge of a tree into an tree additive property; just the kind of distance measure required for the consistency of most distance based tree building algorithms (Felsenstein 1988).

The next step is to look for an interpretation of what -ln[det(Fmn)] is, and how it can be converted into a distance which has an easy interpretation. Since -ln[det(Fmn)] = -ln[det(Plca)] + -ln[det(PtrmPrn)], let's start by removing (as much as possible) the term -ln[det(Plca)] since this is term related only to the base composition at an internal node. If the model is stationary, then we can estimate Pall as the average state frequency across all sequences and so a new distance would be,

dmn = -(ln[det(Fmn)] + ln[det(Pall)]) / r (*)

where r is the number of states (a factor which is explained later)(Notice also that the addition of the final term of the previous equation can also be called a subtraction, since it can be reexpressed as dmn = -ln[det(Fmn)] - ln[det(Pall)]). Unfortunately if the model is not stationary (or due to statistical fluctuations), dmn could be slightly negative since -ln[det(Fmn)] can be less than ln[det(Pall)] when measuring a relatively short distance. A negative distance seems unusual (a bit like negative edge lengths in a tree), but is not serious since the distance is still tree additive. This distance measure was submitted for publication by RodrÌguez and Medina in (1987, pers comm. from Medina), was rejected, and unfortunately did not reappear as part of their 1990 paper (Rodriguez et al.).

One way to guarantee that the amount being taken off by subtracting a term related to ln[det(Plca)] is never more than -ln[det(Fmn)] is to set det(Plca) to its minimum possible value. This minimum occurs when all states have the same frequency, so if there are r states the frequency of each state is equal to 1/r. The determinant of a diagonal matrix with all diagonal elements equal to 1/r is just rln(r), so

dmn = -(ln[det(Fmn)] + rln[r]) / r (*)

(Lockhart et al. 1994, immediately before their equation 3). This distance is indeed additive and always positive, but has the property that it can be positive even if the actual distance between m and n is zero substitutions (again not a problem for tree building, and a big improvement over using -ln[det(Fmn)] since then every tip in the tree then has at least 4ln(4)/2 = 2.77 added onto it when using 4 state characters!).

Another way to remove the effect of the root distribution of character states, and always have dmn equal to zero if the sequences are identical, is to subtract the term ln[det(PmPn)]/2 = 1/2(ln[det(Pm)] + 1/2ln[det(Pm)]) to give,

dmn = -(ln[det(Fmn)] + ln[det(PmPn)]/2) / r (*)

(Lockhart et al. 1994). Lake (1994) gives this same formula (bar the division by r) and calls it a paralinear-distance. Para-linear because while it is not generally equal to the number of substitutions per site, it does keep getting bigger as the number of substitutions increases. Thus there are a whole set of distances additive under the general model described above. All of these are based on the LogDet transform; they differ only in how they attempt to remove the influence of the term , and all may be called LogDet transforms. In general, the term LogDet transform and Para-linear distance have come to be synonymous, since it is this form of the transformation that appears to be most practical in everyday phylogenetic uses. Additionally, it is this last form which is most suitable for estimating divergence times and testing a clock if base composition is unequal (Waddell, 1995).

Barry and Harrington (1987) introduced a distance measure closely related to the LogDet transformations, which is however, non-additive under non-stationary models. They called it an asynchronous distance, and it is calculated as, d’mn = -1/4ln[det(Pmn)] = -1/4(ln[det(Fmn)] + ln[det(Pm)]) (since Pmn = Pm-1Fmn). The problem is that since -ln[det(Fmn)] is the underlying tree additive distance measure, then we are subtracting one quantity, ln[det(Pm)] when calculating dmn, but subtracting a potentially quite different quantity, ln[det(Pn)], to estimate d’nm. This not only leads to asymmetric distances, but also non-additive distances when base composition is not stationary. We have worked examples to show that this in turn can lead to inconsistency of tree selection when a method such a neighbor-joining is applied to just one half of a triangular matrix of these asymmetric distances (see table 3.2 of Waddell 1995). These asymmetric distances are not especially useful, and do not contain any special information (their difference is just ln[det(Pm)] - ln[det(Pn)])(none the less, many findings in Barry and Harrington (1987) are still especially useful and relevant).

An important recent result from Waddell (1995, pp 184-185) is that the LogDet measure of dmn = -1/4(ln[det(Fmn)] + 1/2(ln[det(Pm)] + 1/2ln[det(Pm)]) of Lockhart et al.(1994) and Lake (1994), is equal to the averaged asynchronous distances of Barry and Harrington (1987) (a measure for which there is an interpretation). The average of the asynchronous dmn + dnm is equal to 1/2-1/4(ln[det(Fmn)] + ln[det(Pm)]) + -1/4(ln[det(Fmn)] + ln[det(Pm)]) = -1/8ln[det(Fmn)] + ln[det(Fmn)] - ln[det(Pm)] - ln[det(Pm)]) = -1/4ln[det(Fmn)] + 1/2(ln[det(Pm)] + 1/2ln[det(Pm)]. This result is used next to uncover what distance these equations are measuring.

The next question is just what is the LogDet formula measuring? The answer to this is illustrated next in the case of the 4-state model, but the results extend naturally to any number of states. Barry and Harrington (1987) showed if we think of substitution as being a fine grained or continuous time process, then the asymmetric distance d’nm = -1/4ln[det(Pmn)] is equal to , the expected number of weighted substitutions per site (where the weighting is by the inverse of the proportion of base i measured at the instance the substitution occurs). So let’s imagine there is a substitution from A C somewhere on the path between tips m and n, and at that instance the proportion of base A in the sequence is 0.1, then this adds (0.25 1/0.1) / c (where c is the sequence length) or 2.5 /c to the overall distance measured per site. In contrast the unweighted number of substitutions per site adds just 1/c to the distance per site (the quantity that most reversible model distance transformations estimate).

It then follows that the average of the asynchronous distances, the distance measure of Lockhart et al.(1994) and Lake (1994), dmn = -1/4(ln[det(Fmn)] + 1/2(ln[det(Pm)] + 1/2ln[det(Pm)]) is measuring,

, (equation *)

making this summation in each tiny segment of the path from m to n (Waddell 1995, pp. 101). Thus a substitution from A C (given that pA = 0.1 and pC = 0.4), will contribute 1/8(1/0.1 + 1/0.4) / c = 1.56 /c to the distance measure of Lockhart et al.(1994) (Lake's 1994 distance measure is identical but does not divide by r the number of states).

This new interpretation of the LogDet distance allows us to better understand how it behaves. For example, if all base frequencies are equal, then this weighting will be negated, and the distance measure returned will be the unweighted sum of substitutions per site (noted also by Lake 1994 and Lockhart et al. 1994). Properties that arise from further studies (Waddell 1995, pp. 99-105) include:

If the model has identical site rates and non-equal stationary base frequencies (i.e. not equal to each other but the same in all sequences), the LogDet measure of equation * will, in expectation, return a distance which is greater than the expected number of unweighted substitutions per site. This distance will, however, remain in a constant proportion to the unweighted number of substitutions per site, making it a suitable distance for estimating relative divergence times on a phylogenetic tree.

Generally, the further away from equi-frequency the sequences are, then the larger this weighted distance tends to be relative to the unweighted distance.

If the model is non-stationary, then this LogDet distance is usually (but not always) greater than the number of unweighted substitutions per site (and an approximate estimate of how much so can often be made given estimates of base frequencies at internal nodes).

Under non-stationary models, the LogDet of equation * will often yield distances that are closer to the unweighted number of substitutions per site than any of the standard distance transformations.

This last statement is due to the fact that non-stationary base composition can lead the standard formulae to make widely erroneous distance estimates which are either too large or too small (relative to the true unweighted distance, Waddell 1995 pp. 108-117). Further more, using just pairwise distances, there is not enough information to remove weighting effect caused by ancestral base composition. Indeed, even with non-stationary ML models, this is often difficult or impossible to do (because base composition shifts continuously). Bearing all this in mind, edge lengths on a tree estimated from this LogDet distance can be as, and sometimes more useful, than the edge lengths that would be yielded by standard distance transformations or current likelihood methods (especially when studying anciently diverged molecules like rRNA).

Another way of considering what the LogDet distances are measuring is to look at the similarity between them and the general time reversible distance. If we break the P matrix on each edge of the tree up into a very fine time interval (as the fine grained substitution process suggests is reasonable with real sequences), the as these intervals become infinitely fine, Pi I + Ri, where Ri is the substitution rate matrix in a very short interval, and I is the unit or identity matrix (1's on the diagonal, 0 elsewhere). So the determinant of Pi becomes equal to the trace of Ri (Barry and Harrington, 1987). Recall that the unweighted number of substitutions per site is -trace(PRi), then we can see in another way that the LogDet is 'measuring' what is happening in R in each tiny segment of the tree, but is ignoring the base composition (so giving the weighting of equation *, above). Another way of converging to a similar result under time reversible models is to recall that, dij = trace(Pln[P-1F]), but,

trace(ln[P-1F]) = trace(Rt) = (ln[det(P)]) = (ln[det(F]) + ln[det(P-1])
(equation *)

(Waddell 1995, equation 3.3.1-4) since the determinant of P (= P-1F) is equal to the product of the eigenvalues of P (by the mathematical relationship called the Jacobi identity). The last equation to the extreme right is the LogDet or paralinear distance applied to the symetricised F matrix, so under stationary base composition (a necessary condition for a time reversible model) the LogDet converges to the time reversible distance minus the factor P in the final summation the total number of substitutions (which will given stationarity, at a minimum, see the distance be r times too large, thus explaining the division by r in most LogDet distances). This relationship also has important consequences for applying the LogDet with unequal rates across sites, as discussed later.

Modifying the LogDet to cope with unequal rates across sites

A major limitation of the standard LogDet transform is that is no more robust to unequal rates across sites than other distance measures (Barry and Harrington, 1987)(Lockhart et al. 1994), and unlike time reversible models cannot in itself be modified to be exact under a distribution of rates across sites. It is tempting to think that because d = -trace(PM-1[P-1F]) gives an additive distance with unequal rates across sites and any reversible distance, so to will d = - M-1(determinant(F)). Part of the reason this does not work is that M-1[det(F)] is not equal to M-1[P-1F], except when M-1 equals the logarithmic function. Indeed, since M-1 is (in our applications) always a more non-linear transformation than the straight log function (unless M-1 = ln when site rates are i.r.), and since the determinant is the product of eigenvalues each in the range of zero to one, it follows that attempting to do this will give a number which increasingly overestimates the true distance.

However, Waddell (l995) shows that by removing an appropriate proportion of unvaried or constant sites, or by classifying sites into a limited number of rate classes (just two or 3 is often sufficient), then LogDet transforms applied to each set of sites separately, followed by summation, yields near additive distances under a variety of distributions of rates across sites, including gamma distributions. This dramatically reduces the non-additivity problem arising from unequal rates across sites (additionally, the invariant sites model often fits real functional sequence evolution better than a G distribution). Given that many sequences show evidence of base compositionally change, and clearly unequal rates across sites, and given the surprisingly low variance of the LogDet distances given their generality, then it would seem unwise not to consider them carefully in analysis of medium to large sequence divergences. A good example of the invariant sites-LogDet transform providing new insight was in a reanalysis of the data of Li and Gouy (1989) used to study archaebacterial monophyly. Among a series of six prior hypotheses tested in Waddell (1995) with this data set were the monophyly of the archaebacteria verses the eocyte hypothesis (squares and diamonds in figure *), and the second independent hypothesis of whether Giardia or Microsporidia (or both) constituted the earliest branching in the Eukaryotes (squares, diamonds and triangles respectively). Notice that the LogDet slightly decreases the support for monophyletic archaebacteria, and with invariant sites removed is suggesting the data are poor at discriminating the alternatives (although further analyses with data editing did give clear support for the archaebacteria monophyletic, see Waddell 1995, section 3.8).

Figure *. Impact of the LogDet on analysis of ancient 16S-like rRNA (modified from Waddell 1995, figure 3.11). The x-axis is the proportion of invariant sites removed (according to their own distinct base frequencies). Four different estimators put pinv between 0.2 and 0.3, with all but one in the range 0.25 to 0.3. The y-axis gives bootstrap proportions for trees estimated with NJ. Dotted lines JC distances, solid lines LogDet distances. The lone hollow dots indicate support with 6 parameter iterated ML distances, while the solid lone dots are with LogDet after removing 20% of the most rapidly evolving sites.

In the case of the earliest eukaryotes support for hypotheses shifted strongly after the use of the LogDet. In contrast to the Jukes Cantor, and every other time reversible distance considered, the LogDet did not strongly favour Giardia deepest. Rather, as a an increasing proportion of invariant sites were removed, the data strongly supported Microsporidia first (Waddell, 1995), a result consistent with the fusion of the 5.8s to the large subunit rRNA and corroborated by recent analysis of elongation factor genes. A probable explanation for why removal of invariant sites made such a marked difference, is that inclusion of invariant sites not only result in proportional underestimation of distances, they also can act to 'hide' or 'mellow' base composition differences between sequences resulting in the LogDet not making as much 'correction' as it should have.

Forcing the LogDet to simpler models of substitution

The LogDet method can also be forced to fit some special restrictions of the i.r. and i.i.d. model (Waddell 1995, pp 100-101). One of these is to symmetrize F (by averaging the ij-th and ji-th entries). This corresponds to either a molecular-clock model, a time reversible model, or both. This restriction may be useful in that it will reduce the variance of the LogDet transform (with no sacrifice in additivity if these models indeed hold). A further restriction is to average all entries in F which are expected to be equal under the Kimura 3P model (or its submodels, the 2P and 1P). Interestingly doing this yields an identical distance estimate to the Kimura 3P formula. This is easily understood when it is noted that the log determinate of F is equal to the sum of the natural logs of the eigenvalues of F; and the eigenvalues of F constrained to the 3P model are just 1 ($\ln(1) = 0$), (1-2P-2Q), (1-2P-2R) and (1-2Q-2R) where P, Q, and R are transitions and type one and two transversions (which also helps to explain why the LogDet has such similar variances under these common models, given moderate sequence lengths). Forcing the LogDet to more specific models is analogous to the 'method of moments' estimators for time reversible models developed by Lewis and Swofford (1996). Their more general weighting of the F matrix to meet expectations when P is not equifrequency can also be used with the LogDet. This equivalence also strongly suggests that the LogDet will have a variance very similar to a general time reversible distance (as sequence become longer) when such a model applies. The consequence being that the LogDet need not carry a high penalty in its overall variance, and its general use would not necessarily be an awful thing from this perspective.

Application of the LogDet to many states

The LogDet can be applied to amino acid sequences (e.g. Lake 1994), or even using each of the 61 non-stop codons as states. We expect the variance of the LogDet to be more of a problem the more states are added, thus it may sometimes be useful to group some states together (e.g. into the 6 main amino acid classes). A second problem can also arise; when one or more states is missing (e.g. a stretch of amino acid sequence does not have a cytosine in it) in one or more of the sequences the determinant of both F and at least one P will become zero, yielding a nonsense distance. Possible solutions include removing this state from F altogether, or adding in a small nominal value (such as +0.5/c) and equilibrating the remaining entries in each F matrix (useful if some sequences have a state while others don't). The sampling variance in such instances has yet to be studied. Amino acid sequences also have large differences in rates across sites, so we expect the removal of an appropriate proportion of invariant sites will be useful to improve additivity, and hence accuracy of tree reconstruction. In summary then the LogDet set of distance transforms are additive under much more relaxed assumptions regarding the process of base substitution and allows for the evolution of the base substitution rates themselves. It also tends to have quite acceptable stability to sampling error, given that sequences are at least moderately long (say 200 bases or more) and distances are not huge. Overall this transformation looks set to become a useful and commonly used tool in phylogenetics. However like all methods it makes assumptions about the true process of molecular evolution that we know are violated to some degree. It is important to ascertain whether transformed distances are indeed additive on a tree, and for this purpose methods such as GLS, split decomposition or the distance Hadamard can be useful. It is unfortunate (and quite wrong) if the introduction of a new more general distance transformation is construed as an excuse for biologists to become less critical of their analyses.

16.23 Dealing with 'infinite' distances

Occasionally real sequences and more often the bootstrapping of real sequences will yield a sample F matrix which implies 'too large' a distance. A well known example is where the proportion of sites showing a difference is 3/4 or greater, so the Jukes Cantor distance attempts to take the log of a negative number. In reality, the JC distance is an ML estimator, and the ML estimate of this distance is d as d . The problem is generally becomes worse the more parameters a distance has (except with the Tamura 92 and Tamura-Nei 94 distances which seem especially vulnerable to this problem, explained in the next section).

Rather than setting all such distances to an arbitrarily large number, the default in PAUP* is to set them to twice the size of the largest defined distance for that data set. Logically, the largest distance on any tree cannot be more than twice as large as the next biggest. This seems to work adequately, and in a study (Waddell, 1995) the exact value these undefined distances were set too did not seem to critical with NJ. With time reversible distances, undefined distances can often be avoided by taking the distance to be just the sum of the logarithm of the eigenvalues which are always positive for that distance (equivalent to a scalar of the transversion distances for the Kimura 2 and 3 ST models). An alternative is to map the data to a set of character states which change more rarely (e.g. amino acids or perhaps just AG verses CT).

The sample error variances of distances from sequences:

16.24 Time reversible distances

Estimating the variance of time reversible distances usually proceeds via the statistical approximation known as the 'delta method'. Take, for example, the Jukes-Cantor distance d = -3/4ln(1-4/3p) (where p is the proportion of nucleotides different). The variance of p, as already explained, is 1/c(p(1-p)). The multiplication by the constant 4/3 increases the variance by 42/32 or (16/9)1/c(p(1-p)). Next comes taking the logarithm of (1-4/3p). Here the delta method is applied, which is based on the square of the gradient of the transformation curve at the point the value being transformed. The gradient of the curve ln(1-4/3p) is equal to 1/(1-4/3p), so the variance becomes, (16/9)1/c(p(1-p)) / (1-4/3p)2.

Lastly, multiplication by 3/4 reduces the overall variance by 32/42. So canceling with the previous term due to multiplication by 4/3, this leaves the total variance of d as,

Var[d] = , so s[d] = .

With more complicated distances, there is more than one variable, so the covariances between variables need to be taken into account in estimating the overall variance of d. For example, the Kimura 2P distance has the variables P and Q, so their variances (i.e. P(1-P)/c and Q(1-Q)/c) and covariance (-2PQ/c) both figure in the final variance which is,

Var[d] = ,

where a = 1/(1-2P-2Q) and b = 1/2 [1/(1-2P-Q) + 1/(1-2Q)] (Kimura 1980).

Unfortunately, many authors who have derived variances for closed form distances (e.g. Tamura 1992, Tamura and Nei 1994), have not incorporated the terms in P, which are integral to the distances when P is not assumed to be equi-frequency. This in turn leads to inconsistent estimates of the variance of these distances, degrading their value when used with tree estimation methods such as WLS. Generally, the published delta method variances which do not take into account the variance in P lead to overestimates of the true variance as the sequence length increases .The variance of the URAS-GTR distance (equation *) is given in Waddell (1995) and Waddell and Steel (1996) as an extension of the variance given by Barry and Harrington (1987) and this does take into account all the interactions of the 9 free variables in F. This 'BHaWS' variance is,

(*)

where (Gkl) are elements of the matrix, G, B = I - P (where Bt is the transpose of B), while P = P-1F, R = M-1[P] (replaced by their estimates when working from a finite sample) and c is the sequence length. The term which changes with the distribution of rates across sites is ar. This term is given by the coefficients of the Taylor series expansion of M-1[1-x] = -. For example, in the case of the i.r. distribution, ar is the coefficient of xr in -ln(1-x), and so ar = 1/r (i.e a1 = 1, a2 = 1/2, a3 = 1/3, ...). For the G distribution with shape parameter k the series becomes, ar =

[(k+1)(2k+1)...((r-1)k+1)]/(r!kr-1), for example, with shape parameter k = 0.351, a1 = 1/(1 0.3510) = 1; a2 = (0.351+1)/(2!(0.3511)) = 1.92; a3 = [(0.351+1)(0.702+1)]/(3!(0.3512)) = 3.11. As k goes to infinity, this series converges to that for the i.r. distribution, as expected. For the inverse Gaussian distribution, ar = . The main computational concern is to take sufficient terms in estimating G, often 20 will be sufficient for accuracy to more than 4 places. If invariant sites are being compensated for (either alone or in combination with variable rates across sites) the most flexible way to take this into account when estimating the variance is to first subtract Pinvpinv from F (where Pinv and pinv are the assumed base composition of the invariant sites and total proportion, respectively), then multiply each entry in this new F matrix of variable sites by 1/(1-pinv) (Waddell and Steel 1996). Following this, evaluate the variance with the BHaWS formula (Eq.*), remembering to modify c (the sequence length) by multiplication by (1-pinv), since the distance is now for just the variable sites. Note, that this estimate of the variance does not take into account the variance expected due to estimation of pinv, which will act to inflate the overall variance.

Fortunately, there is a simple solution to deriving exact delta method variances for all time reversible distances, with or without unequal rates across sites. Combining the BHaWS variance with the weighting function of Swofford and Lewis, gives a delta method variance for any closed form reversible distance when base frequencies are unequal taking into account all the interactions of variables. All that need be done is that F is replaced by F* the weighted matrix, as too are all derivatives of F appearing in the BHaWS formula (i.e. R, P, and P). The consistency of these variance estimates have been checked with simulations and can also be shown by algebraic manipulation.

The finding of Waddell (1995) that the Tamura and Nei (1994) and sometimes the (Tamura 1992) distances have higher variance than the GTR distance is also illustrated by the BHaWS formula. It would normally be expected that a distance with fewer parameters would give a lower variance than one with more. However, this has been seen not to be the case, and the explanation offered by Waddell (1995) is that the GTR distance is an ML estimator (for a proof see Waddell 1996), while the Tamura-Nei and Tamura distances are not (e.g. Zharkikh 1994). Thus they 'blow-up' errors more than the GTR distance. If you take a valid F matrix and apply the BHaWS formula, then form F* and reapply the formula, you can see the increased variance appear. Thus, our recommendation is that even when the Tamura or Tamura-Nei model's assumptions seem reasonable, one should defer to the URAS-GTR distance, or else use iterated ML distances, where the reduced parameters of the simpler models will reduce the variance.

A similar problem also crops up with iterative ML distances, based on reversible unequal frequency models. These calculations tend to also ignore the possible fluctuations in P, by treating these as fixed. If entries in P are optimised also, then the variance can be derived by estimating the information matrix for each iterated ML distance, and estimating the total variance as the sum of the variances and covariances of entries in R derived from the information matrix (the information matrix is also easily accessible if R is fixed for a set of sequences, and estimated by ML). If R is estimated in some other way (e.g. as a sum of the result of applying M-1(P-1F) to many pairwise comparisons) its variance-covariance matrix is more difficult to define (it even requires the correlation structure of the tree linking the sequences to be considered).

16.25 The variance of iterated ML distances

It is possible to get an approximate estimate of the variance of dij estimated by this ML method. That is, after the initial maximisation of lnL increase the estimated distance, dij, until lnL decreases by 1 and store this increment in dij as x. Then decrease dij until lnL again drops by 1 from its maximum (decrement, y). Estimate the variance as 1/2(x2 + y2). This basic approach is standard and analogous to Felsenstein (1993) who uses it to estimate of the variance of an edge in the ML tree (program DNAML). As with the previous delta method variance, this estimate of the variance ignores the uncertainty in knowing R and M-1 (i.e. site rate distribution and shape parameters). These additional factors are not presently considered, but can and should be incorporated into a bootstrapping routine (with re-estimation of all variable parameters on each replicate).

16.26 The variance of the LogDet and its statistical properties

Barry and Harrington (1987) derived the variance of the asynchronous distances, and following the same procedure Lockhart et al. (1994)

derived the variance of -ln[det(Fmn)]. Accordingly we can give the variance of the ”rln(r)” (i.e. LogDet reexpressed in matrix form as,

Var[-1/r-ln[det(Fmn)] + rln(r)] = (trace(BFmn)-r2) / (r2c),

where B is the matrix whose ij-th entry is the square of the ij-th entry of F-1 (the inverse of Fmn), and r is the number of states (since rln(r) is a constant it does not figure in the variance).

It is also possible to derive an expression for the variance and covariances of the other LogDet measures which subtract observed base frequencies (themselves random variables). Barry and Hartigan (1987b) give the variance of the asynchronous distance, and it can be rewritten in matrix form as,

Var[-1/r-ln[det(Fmn)] + ln[det (Pm)] = Var[-1/r-ln[det(Pmn)]] = (trace(BPmn)-r2) / (r2c),

where B is the matrix whose ij-th entry is the square of the ij-th entry of P-1 (the inverse of Pmn = Pm-1Fmn), and r is the number of states. Recalling that the LogDet distance is the average of paired asynchronous distances, and noting that Var[x + y] = Var[x] + Var[y] -2Cov[x, y] we can derive the variance of dmn = -(ln[det(Fmn)] + ln[det(PmPn)]/2) / r as,

Var[dmn] = (trace(BPmn - X) / (r2c) (*)

where X is a diagonal matrix with non-zero elements equal to (pmipni)-1/2. Notice that trace of X will be 1 if both sequences are equi-frequency (thus giving the same variance as eq. *), greater than 1 otherwise, and increasingly large the more the base composition of the two sequences differs. Accordingly, the variance of this form of the LogDet will be minimal of those considered. However, this does not improve its tree estimation properties, since all the LogDet distances are additive in expectation, varying only by pairwise factor (like clipping the tips of the true tree), something that many tree selection algorithms (e.g. GLS, WLS, NJ, four-point metric) are insensitive too, but which will affect the FM algorithm (since it equates the relative variance of distances with their relative size). An interesting property of the above formulae, is that when evaluated analytically they return the same values as the delta method variance of equi-frequency distances (e.g. the Jukes-Cantor, the

K2P). Thus under simple models the LogDet transforms are expected to have variances which converge, with increasing sequence length, towards those of at least some of the standard distances. In addition simulations by the authors (Swofford, Lewis and Waddell 1995, Waddell, Swofford and Lewis 1996), show that this relatively ”low variance” feature results in tree building being as reliable with LogDet transforms as with other ”stationary” distance transformations as sequences become longer (there is no magic number at which this occurs, but for many models, sequences of length 500 will often do it). Bootstrap resampling of real data using the programs Trees (Penny et al. 1993), and PAUP (Swofford 1995) tend to confirm that the stochastic stability tree selection, following LogDet transformation, remains comparable to that with other distance transformations, even when studying deeply diverged rRNA and protein sequences (using first and second positions, Waddell 1995).

The variance of the invariant sites-LogDet distance is calculated in an equivalent way to that for the URAS-GTR distance, i.e. premodify F by subtracting Pinvpinv, then multiplying by 1/(1-pinv), and modifying c by the factor (1-pinv) (Waddell, 1995).

There are other ways of estimating the variance of distances, such as Monte-Carlo simulations, or the bootstrap. These methods introduce additional sampling error when applied to data which is already a sample, and due to the power functions used in these genetic distances inflating sampling errors (e.g. see Waddell et al. 1994), this sees these resampling routines biased towards overestimating the total variance. In contrast, the delta method applied to samples can return nearly unbiased estimates of the variance (Waddell et al. 1994). Additionally, they tend to carry more computational burden then the delta method approach, but allow a greater degree of flexibility and some freedom from the need to make limiting assumptions.

16.27 The impact of variance and bias in a realistic situation

The variance of pairwise distances, and their mean deviation from additivity (bias) are two main factors influencing the reliability of tree recovery. If either moves too far from optimal, the distance will become distinctly less reliable for tree estimation. As mentioned earlier, both factors together sum up to deviation from the expected distance, and this deviation can be measured as the RMSE (Root Mean Square Error), or more specifically .

To examine these factors, let's consider the distance from an African ape (human) to a lesser Ape (siamang) based on the mtDNA of Horai et al. (1992). Let's start by estimating then fixing the rate matrix and distribution of rates across sites. Ml estimations of the overall likelihood of the model and distribution shape parameters are shown in table 3.

Table 3 ML estimation of rates across sites and R for Horai et al. (1992) Hominoid mtDNA sequences. The G distribution is approximated by 10 rate categories.

Model Incl. orangutan Excl. orangutan lnL parameter est. lnL parameter est. Invariant sites 14427.0 pinv = 0.618 Gamma dist. 14426.4 a = 0.245 Invariant sites plus Gamma dist. 14423.3* pinv = 0.544 a = 1.69

Matrix R for the Invariant sites - Gamma model. Delta-method variance, bias2 and RMSE for some of the 'direct transform' distances in PAUP* (based on parameters recovered by ML in the previous table).

model site rate variability distance i.r. pinv = 0.618 G, a = 0.245 pinv = 0.544, a = 1.69 GTR TN HKY T92 F81-TN K3ST K2ST JC

David: I will complete this table if you like it, else, it could be the first bit for the chop to shorten the chapter.

How to estimate the distribution of rates across sites

As we have already seen (e.g. in figure 3) distance estimates can be highly sensitive to the distribution of rates across sites. Indeed, in evaluations, of distance methods relative to ML, there are suggestions that distance based methods are more sensitive to this factor, with regard to the efficiency and possible inconsistency of tree selection (e.g. Waddell 1995, Lockhart et al. 1996, Yang 1996, Waddell et al. 1996). It is appropriate then to consider this factor in detail in this chapter.

When estimating the distribution of rates across sites, there are two main decisions. Firstly deciding on a distribution of sites rates to use, and secondly how to estimate its shape parameter(s). The first question is analogous to picking a model e.g. like specifying the form of R. The second is analogous to finding the optimal values for R. Here fit measures, like likelihood, are useful for both steps. Apart from ML there exist other methods which are valid, quicker to compute and possibly also more robust with some types of real data. These are mentioned secondarily.

16.28 Estimating distributions and their shape with ML

If distances are the only data, then the ML method, as described earlier is an iterated generalised least squares method. Unfortunately, no such method is currently implemented. It is possible to use, instead, ML on sequences, which is implemented for many of the models for which distances are also available. PAUP* offers simultaneous optimisation of many parameters in the models, including edges lengths, rates in S, a proportion of invariant sites and / or a G distribution. While ML on sequences is a useful method, it does not directly optimise additivity of distances. Here we arrive at a conundrum. Hopefully, the major difference between the parameter estimates by ML on sequences and ML on distances will be that the former will be both more statistically efficient and robust. Additionally, hopefully using parameter estimates closer to their true values will yield distances which are more useful at identifying the true tree (although this is something to be tested).

It is important also to understand how a shape parameter may change depending on the nature of the model chosen, including the tree. Figure * shows a summary of the main interactions known to date. Firstly from the figure it is clear that the shape parameter is skewed towards more extreme values (i.e. implying more unequal site rates) the worse fitting the tree is. This factor is recognised in the studies of Waddell (1995, figure 5.3) and Sullivan et al. (1996). It contradicts the claim of Yang (1993) that it is sufficient to evaluate the shape parameter on the star tree. Indeed, this appears to often be the worst tree on which to evaluate the shape parameter when the data has considerable phylogenetic structure (as one would hope it does). The rule must be: evaluate the shape parameter on the best tree available, and if there is doubt, consider its value also on alternative likely trees.

Figure 4. The effect of model on the estimation of shape parameters, a (squares) and pinv. (diamonds) (a) The effect of the form of R (b) The effect of the tree. Data mtDNA sequence from Horai et al. (1992).

An explanation for worse trees having more extreme shape parameters is how likelihood tries to explain excess homoplasy. A worse fitting tree has internal edges which do not coincide with the best supported internal edges. Rather, these site patterns, which may really be synapomorphies must be explained (shown to be more likely) as (often frequent) convergences and parallelisms. A number of ways exist to do this. Making edges (especially particular external edges) longer can boost the expected frequency of parallelisms and convergences, as more unequal site rate distributions can also do (e.g. very rapidly evolving sites can imply all site patterns have the same frequency). Sometimes a higher ti / tv ratio can have a similar effect (although this factor is not so potent since it can run into conflict with the overall ratio of observed transitions and transversions). On data sets with poorly defined hierarchical structure (or not fitting the model well) these trends may not show up clearly (Waddell 1995, figure 5.6) and this may have lead to Yang's (1993) claim.

Figure * also shows that the substitution model chosen can have a profound effect on the estimated shape parameters. Thus far, the trend seems to be that if the model ignores a major feature like unequal base composition, or unequal transition to transversion rates, it will underestimate the unequalness of site rates. Allowing extra parameters does not compensate for this (e.g. the generalised Kimura 3ST model used in Waddell and Penny 1996) and if the situation was reversed and the rate matrix really was varying from edge to edge the same might hold when using any time reversible models. A useful rule is to aim to evaluate the shape parameter under the model which explains most e.g. the model with highest AIC (AIC = lnL - (no. parameters in the model)). However, it is also equally true that if the process of evolution is distinctly unlike the model, then it is unclear which, if any model will offer the best estimates, and all models need to be treated with suspicion. The best suggestion appears to be to consider the range of values being predicted, and its possible effects upon the parameters of most interest (most often tree topology, but also other parameters such as edge lengths, ti / tv ratio etc.).

It appears form figure * that use of a worse fitting model generally underestimates the shape parameter. Notice, however, if the details of the model are only slightly wrong (e.g. a comparison of a K2 to K3 model) the reverse trend occurs to a slight effect. Overall, the conclusion must be that if you wished to make a best estimate of a distribution shape parameter, you need to get major features of the model correct. This is presently most feasible with closely related (e.g. diverged in the last 20 million years) sequences where sites are neutrally evolving. For very old sequences (e.g. over 100 million years of divergence) where evolution cannot be i.i.d. we should be prepared that our estimates of the unequalness of site rates could be out by a factor of two or more.

A simple test of the significance of allowing for site rate variability is to take the ratio of the lnL statistic with it and without it, and under the hypothesis that the difference is due to random fluctuations, it will asymptotically be distributed as c2 with d.f. the difference in the number of parameters (e.g. pinv or G, 1, for pinv + G, 2).

16.29 Dangers when using approximated G distributions

Lastly, the above discussion was assuming that we were using an exact method of calculating likelihoods for the G distributed model (e.g. the method of Steel et al. 1993, Waddell and Steel 1996). However, in practice, PAUP* uses an approximate method to achieve the integration of a G (Yang 1994, Waddell 1995), and this can give rise to problems. In figure *, the fit of model and estimated shape parameter a are plotted as the G is approximated by x rate classes, with each rate being the mean rate for successive fractions of 1/x of the sites drawn form a G distribution. This approach can be thought of as either a new distribution (Yang 1994) or a crude integration (Waddell, 1995). Either way, several issues are raised.

Firstly, many distributions could give rise to a nearly identical discretised distribution, so the link to a G is tenuous with few rate classes. This makes it difficult to decide if the data are really favouring one type of distribution over another (Waddell 1995, Waddell et al. 1996). If we had, for example, used just 4 rate classes to achieve the integration, we would have concluded the G fitted significantly better than the invariant sites model, something incorrect when a better approximation is made.

Secondly, the shape parameter may have a tenuous link to the true G distributions shape parameter, here being massively incorrect with just two rate categories, but still decidedly biased with a moderate number. In this case, by 15 rate categories things are settling down. Because this bias is towards overestimating a, there is the danger of 'long edges repel' in the anti-Felsenstein zone, as distances are biased towards increasing overestimation. To safeguard this danger, we revert to advocating considering the important biological questions with a variety of shape parameters spanning the likely range.

Thirdly, if a discretised distribution is fitting best with fewer categories, this has interesting implications. One is that the tail of the continuous distribution is not necessary, and may be a major cause of the decrease in fit as more rate categories are used. This is because with real sequences, we expect a fixed maximal rate of substitution, which in the case of mtDNA should be near the rate of the four fold degenerate sites. Continuous distributions like the G, however, go on predicting ever faster sites (Waddell et al. 1996). Alternatively, the intervals of the discretised G may just happen to coincide with some unexplained discreteness in the actual data (less likely here, since the change is smooth with increasing number of classes). The closest approximation to the G fits nearly 5 lnL units worse then with 5 rate categories, and this ironically, provides good evidence that the true distribution is not G (or for that matter invariant sites only). Clearly, we should view our estimates of the distribution of rates across sites with a healthy skepticism for the finer details.

Note lastly, the difference between a few and many rate categories may not only affect the shape parameter it is also potentially distorting other quantitative (e.g. the rank of trees) or qualitative aspects of the analysis. The latter include the overall rate matrix, R or S, and sometimes to a substantial degree relative and absolute edge lengths. Such strong biases should be checked for by running calculations with at least two setting of the number of categories (e.g. 4 and 10). PAUP* offers the option of also using the median, rather than the average to represent each rate category. In our experience this option often: slows considerably the convergence of parameters to their optima; may sometimes result in a better fit of model to data (e.g. approximately 1 lnL unit better in the case of 10 rate categories and the Horai data); results in an inability to converge (e.g. negative eigenvalues in the rate matrix for less than 9 rate classes, Horai data), and tends to bias the shape parameter further towards zero. Be aware of this if you chose to use it.

Figure 5. The -lnL and the estimated shape parameter changing as the number of rate categories used to approximate a G distribution range from 2 to 70 (GTR model, Horai et al. 1992 data).

Other estimators of site rates

Here we consider how to estimate the parameters relating to unequal rates across sites for use when making distance transformations. There are a variety of methods to make the estimation of the proportion of sites which may be treated as invariant. Promising alternatives are:

Capture recapture (or Petersen) statistics

Capture recapture statistics are often used to estimate the population sizes of wild animals. The basic idea is that you mark n1 animals in a population and release them. You then take a sample some time latter (and you can assume this sample is an independent random sample of the whole population), and the proportion of marked animals in this sample gives you information on how big the population is. For example, if 100 animals are marked and released, then in the second capture of n2 = 200 only 40 of these have marks on them (m), then our estimate of the total population is the number in the first sample, times the ratio of the number in the second sample divided by the number marked (m), i.e. n1n2/m = 100 200 / 40 = 500.

This simple statistic is surpassingly important and well studied (Seber 1982). If we separate sequence sites into a first sample with some 'mark', then take a second random sample and look for this mark an estimate of variable sites in these sequences can be made. For protein sequences, Sidow et al. (1992) used sites showing any change (in a set of t aligned sequences) in the first position as n1, sites showing any change in the second position n2, and the number of codons showing changes at both first and second positions (m) to indicate the proportion of variable codons pvar so pinv = 1- pvar. Waddell (1995) applied the same principles to all sequences by counting any changes at a site (n1) in one set of sequences that shared no overlapping history with a second set of sequences. In this second set, the number of sites showing any change in the aligned sequences is n2, while sites showing change in both sets are designated m. A suitable way to define either n1 or n2 is that it is a monophyletic group with respect to the other group. In all cases the variance of this estimator is calculated following Seber (1982). As long as n1, n2 and m are all sufficiently large (see Seber 1982), the distribution of the statistic is approximately normal and we can use this to test that pinv = 0.

An important property of 'capture-recapture' is that it gives an ML estimate under some minimalist assumptions of the nature of the model of sequence evolution, and so can be used as a check against the predications of more specific estimators (and the two ways of capturing and recapturing with codon sequences, also provide a check against themselves).

16.30 Fitting models to the parsimony reconstructed number of changes per site

If all sites had a single rate, then the number of changes per site would be expected to follow a Poisson distribution. If sites fell into two other wise i.r. rates classes, then the distribution of changes per site will be a weighted mixture of two Poisson's (see Fitch and Markowitz 1970). If site rates follow a gamma distribution, then the distribution will be a mixture of many Poisson distributions, with the weighting (relative frequency) of each being related to the density of the gamma distribution (i.e. an integral). In the case of the gamma this integral has a closed form solution, which is the negative binomial distribution (Uzzel and Corbin 1971). Any other distribution of site rates will also be an integral or summation of some sort. (and if not in closed form it can readily be calculated with numerical methods). The major shortcoming of this method is that it is difficult to estimate the number of changes per site. Parsimony offers one approach. It's major drawback is that it will be biased towards underestimating the number of changes per site, especially for the most rapidly evolving sites. Hopefully, this will have only a minor bias towards underestimating the unequalness of site rates (Wakeley 1993).

So in order to estimate the total number of changes, it remains to match the inferred to expected changes. This can be done with any number of fit procedures. Good examples are the X2 (Fitch and Markowitz 1970) or G2 or ML (Waddell, 1995) statistics. The solution can easily be found for any distribution of rates across sites using optimisation methods such as Newton or conjugate gradients (e.g. a mixture of two gamma's plus invariant sites, Waddell 1995). In some special cases the solution to finding the best fit can be written in closed form. These include a method of moments estimator for the G distribution (which is expressed through a negative binomial distribution of site changes, Uzzel and Corbin 1971) or a likelihood (G2) solution for the same (Sullivan et al. 1995). These give a minor reduction in computational cost, but unfortunately reduce flexibility (in terms of both the criterion of fit and the distribution of site rates). In all cases, the fit statistic G2 or X2 is asymptotically c2 distributed allowing a test of alternative hypotheses about site rate distribution (e.g. Waddell 1995, section 3.6.4).

This basic method can be improved by using a models properties to better infer the likely actual number of substitutions per site, rather than just the observed number.

16.31 Marginal Likelihood Estimates

The parsimony reconstruction of the number of changes per site is biased downwards, especially for sites with large numbers of changes. Another way to get an estimate of the distribution of rates across sites is to use the model to predict how many sites should show 0, 1, 2, 3, ..., t substitutions per site. Then, measuring the fit of these to predicted counts with the observed counts, the models assumptions are modified to allow different distributions of rates across sites (here again, if the likelihood ratio statistic is used to measure fit, then the AIC criterion offers a rule of thumb as to which model best describes the data). While computationally expensive (like ML), a desirable aspect is that that the fit statistic (e.g. X2 or G2) will often have near asymptotic properties (since cell counts will often be high). This conformation to the c2 distribution can be improved by grouping the cells expected to be represented by less than five sites.

16.32 Why it is not wise to use the ME score or the FM SS to optimise a or pinv

It is tempting to use the ME (OLS) fit statistics (the OLS SS) or that of the FM method to optimise distribution shape parameters such as a or pinv. Unfortunately this often does not work well or can be very misleading with the sequence lengths commonly used. The central problem is that as a distribution of site rates becomes more extreme (e.g. a 0, pinv 1), then the variance of distances rise, sometimes dramatically if the site rates are far from equal (e.g. $a < 1$, $pinv > 0.2$). This in turn means we will often expect the RMSE error to increase as variance will often grow faster than the decrease in bias2. If this occurs then the unweighted (OLS) SS will just keep getting worse as site rates are assumed more unequal. Only as sequences become longer, is the improvement in additivity expected to have a larger effect on the overall sum of squares than the increasing variance. Even then, the if the increase in variance is not taken into account the minima for the SS will often be biased towards more homogeneity of site rates.

The crude ME fit statistic, the sum of edge lengths in the tree, might also be contemplated for use. While it may be used to compare two trees (Rzhetsky and Nei 1992), it is a statistic nearly wholly dependent upon the magnitude of the data. That is, if all distances are multiplied by two, it will become twice as large. As such, it is not useful for direct optimisation of shape parameters since it well known that unequal rates across sites make distances larger in expectation (e.g. Golding 1983, Waddell and Steel 1996).

WLS counters the increasingly worse fit due to escalating variances, but does not take into account the changing correlations of transformed distances as the distribution of rates across sites changes. These often strong positive correlations can have a distinct effect on the overall sum of squares (e.g. see figure 3. of Waddell 1995). Here, failing to take into account these correlations sees the residual SS getting worse more quickly than it should. The result is a distortion of the shape of the curve of SS plotted against shape parameter from being smooth and parabola like to diving to the minima more suddenly then expected, then rising again very rapidly. The FM method makes a crude guess at what the variance of a larger distance will be. Depending on whether this guess is fight or wrong the FM SS can lead to either dramatically over or underestimation of shape parameters. In summary then, in most situations these simple sum of squares statistics should not be trusted without further detailed inquiry.

16.33 Separating transition from transversion distances and mapping down to 2-state distances

The general time reversible distance returns a total of six distinct amounts of substitution in the matrix PRt, which a may be considered a ”corrected” F matrix (or matrix S). These six numbers can be separated and studied in various ways (e.g. Lanave et al. 1984, Waddell and Steel

1996). This separation is always valid if the model is time reversible. One way we have found especially useful, is to build at tree from each distinct type of substitution event, e.g. sag up to sgt. This can do two things: if base compositions are equal, then this method will give consistent estimates of exactly how many substations of each type occurred in each edge of the tree (even if the form of R changes, but P remains the same, so the model is not necessarily time reversible, see Waddell and Steel 1996). Normally though, as R starts to change, so P accordingly changes making these separate estimates no longer exact (except in special cases, e.g. see Hendy et al. 1994). However, if amounts of evolution are not large the estimates will still be approximately correct. and can still give a clear indication of shifts in the substitution pattern e.g. see Waddell and Steel (1996).

These same restrictions of homogeneity of R and stationarity of P also apply with more specific restrictions of R. Such examples include the separation of transitions from transversions to give two distances under the TN93 model (e.g. Tamura and Nei 1993, Kumar et al. 1993). Only in special cases, like the Kimura 3ST model, can transitions and transversion rates vary independently and still give exact distances (e.g. ask Mike H for ref.).

There exist some special cases where it is possible to map four state characters down to just two states (e.g. R/Y) and still get consistent distances. In these cases, the equation,

d = bM(1-dobs/b) (*.*)

may be used on the two state data (where b = 1-R2-Y2 and M reduces to the ln function when sites have equal rates). This result is a special case of the 'groupability' of Markov chain processes e.g. proposition 5.9 of Iosifescu (1980), Waddell (1995: pp 83, pp 162-163). If we characterise the transition matrix on each edge of the tree as, P = . Then states 1 and 2 vs 3 and 4 are groupable without distortion of the model if, b + c = e + f, and g + h = j + k (giving P at most 10 free parameters, since * is one minus the other entries in that row). Interestingly evolution by the generalised Kimura 3ST model will always meet this condition, but a stationary time reversible model need not. The condition for evolutionary parsimony to be consistent is similar, except we replace the constraint on sums of entries with b = c, e = f, g = h, and j = k (known as the balanced transversion model, e.g. Navidi and Beckett-Lemus 1992). Consequently, while both procedures place emphasis upon transversions, they are guaranteed to be consistent under slightly different models.

Note that even the LogDet must meet the conditions for ’groupability’, so that when reducing the data to just two states and applying the 2-state LogDet a distortion of distances away from additivity would be generated if the 4-state data was expected to give additive distances with the LogDet. In practice, this possibility of increased bias may be minimal with such reduced distances being more additive than their 4-state analogue when working with large distances. An equivalent argument holds for any number of states (e.g. when using amino acids as states).

Consequently, PAUP* always estimates transversions distances from R (with the appropriate constraints on F), so that groupability is not a direct issue unless you are using an equate command.

16.34 Distance estimates from amino acids

All of the basic theory we have outlined above can be applied directly to amino acid sequences, or going one step further to the finer resolution of the 61 non-stop codons (e.g. Muse and Gaut 1994, Yang and Goldman 1994). Using this larger number of states eliminates the effect of correlations between site changes due to the genetic code, which are inherent (but of undefined importance) when estimating distances from coding nucleotide sequences. They can also be applied to other situations where it is useful to consider the state to be a discrete combination of nucleotide sites. One such example being pairings such as GC or AT which predominate in stem regions of rRNA, and can be considered to give rise to 16 distinct states (i.e. AA, AC, ... , TT) (e.g. Schrˆniger and von Haeseler 1994). In all cases the basic theory of time reversible models and LogDet distances applies also.

The major problem with assuming models with more states is that the data becomes more sparse. This means there are fewer data points relative to the number of items being estimated. With short to moderate nucleotide distances, this often leads to the variances being relatively larger compared to the estimated distances (especially since grouping sites into single states reduces the effective sequence length). However, if nucleotide sequences are approaching saturation, distances based on amino acids may well drop the effective variance, and equally importantly reduce the sensitivity to systematic errors. With amino acid sequences, it appears that both the chemical properties of amino acids and the underlying genetic code tend to be dominant factors in determining the probability of a substitution. Counting these amino acid substitutions from many different closely related proteins, allowed Dayhoff and colleagues to estimate the average relative rates of the 380 possible changes, the so called Dayhoff matrix of Dayhoff et al. (1978). This matrix has recently been updated by Jones et al. (1993) and a specialised form for mtDNA encoded proteins inferred (Adachi and Hasegawa, 1996). These are in essence large average F matrices inferred using either parsimony (Dayhoff et al., Jones et al.) or ML (Adachi and Hasegawa) on estimated evolutionary trees of related sequences. If such a concatenated F is considered to contain very few multiple hits (e.g. Dayhoff et al.) then P is estimated as P-1F, Rt is just P with diagonal elements replaced by minus the sum of the row, and S = P-1R. (Note: S has (400 -20)/ 20 = 190 distinct elements, while R is made up of 190 + 19 from P). Now trusting that the substitution process in all proteins (and all positions) is generally governed by similar physico-chemical rules, a total distance of -trace(PRt) is made after fitting Fobs to Fpred (= Pexp(Rt) = Pexp(SPt)) altering just parameter t. Felsenstein (1993) makes the fit parameter G2 or ML, while it could equally be the minimum X2 statistic (the residual fit would be illuminating for assessing the validity of the assumptions of this procedure). It is possible to add 19 parameters to the distances fitting parameters by substituting in P from the sequences from which distances will be estimated (as this can vary substantially from the average of those used to estimate R). This is done in MOLPHY (Adachi and Hasegawa 1996), and will be offered in future versions of PAUP* (along with unequal site rates).

If divergences not large, then some of the simple time reversible models may be equally useful. Two often used examples are the 1 parameter Poisson distance, d = -19/20 M(1- 20p / 19)), or the 19 parameter proportional model, d = -b M(1-p / b) (i.e. the 20-state analogues of the Jukes-Cantor and F81-TN84 distances). Here M reduces to the natural log function if site rates are assumed identical (else see section * earlier), p is the proportion of amino acids different in the two sequences compared, and b = 1 - Spi2.

Which distance to use?

This is a difficult question to answer, as it is answered in a number of ways. In a direct analogy to choosing an ML model by a criteria such as that of Akaike (e.g. Kishino, et al. 1990), one could seek to minimise the SS error of the optimal tree from the transformed distance data, adding a penalty of 1 for each additional parameter allowed in the model. Unfortunately, GLS methods are not available. Even if they were, there are alternative arguments.

It can be argued that the AIC or related model selection criteria are not the ultimate indicator of the best parameters to use to estimate the tree. They indicate which parameters have a place in the overall model, but if we are more interested in only a part of the whole collection of parameters, the answer could be different. What if we wished to sacrifice niceties of the model such as edge length estimates, and reduced the number of parameters to also reduce statistical fluctuations in parameters deemed more important? In some parts of the parameter space this can work. Examples are the simulations of Saitou and Imanishi (1989) and Waddell et al. (1996). In each case a simpler distance results in higher rates of recovery of the true tree (in the latter case, even considering all forms of tree, e.g. Felsenstein and anti-Felsenstein as equally likely).

What emerges, however, is that this boost in the efficiency of tree selection with the simplest distances is limited to parts of the tree space, generally, the less extreme parts. Highly unequal lineage rates, high ti / tv ratios or shifting base composition, for example, all tend to nullify any advantage of the simpler models. Further, it is still unclear how large the advantage really is, as most evaluations have been on only a few taxa with a few fixed tree topologies. Additionally, use of iterated ML distances (especially with fixed rate matrices), can certainly close up the advantage of using simpler distances, especially when transformed distances with proportionally high variances (e.g. those of Tamura 1992 and Tamura and Nei 1993) are weeded out and replace by iterated ML estimators.

Another factor comes into it also, and that is do simpler distances give rise to reasonable estimates of the statistical support for a tree? As discussed earlier, more parameterised distances tend to show potentially higher levels of bias due to the non-linear logarithmic transforms. This may reduce bootstrap support on many trees by increasing the sampling variance of the distance (although it may boost bootstrap support on the Felsenstein type of tree due to a long edges repel effect). Again this factor is considerably reduced by using iterated ML distances, as these act to reduce the overall variance of the distance. However, conversely, use of too simple a distance will make the distance estimates appear more solid than they really are, and this can give rise to bootstrap supports which are too high, especially in cases where long edges are attracting and the optimal tree places such edges together. In many cases though, choice of any of the more sophisticated time reversible distances (e.g. those allowing unequal base frequencies and separate ti and tv rates) sees a very similar tree and similar bootstrap support. The most substantial changes often occur when the invariant sites-LogDet transform is used. An example is furnished earlier in the case of the position of the Microsporidia based on 16S-like sequences. Since studies are suggesting that base composition is fluid, even within a group such as mammals (or in the example given, within great apes even), it seems wise to at least consider the implications of a LogDet analysis. Equally, the effect of invariant sites is often overlooked, nowadays somewhat unforgivably. Accommodation of invariant sites and / or unequal rates across sites will often see bootstrap support decrease on many nodes, often justifiably. However, this need not always be the case as the example with Microsporidia shows it is only mixing invariant sites with the LogDet that a clear branching order emerges.

16.35 Using distance methods to estimate edge lengths

If you have a tree and are going to use distance based methods to estimate edge lengths, say for divergence time estimates, then be a little cautious. The ideal way to do this would be with iterated GLS (e.g. Hasegawa et al. 1985). Of the methods available in PAUP* presently, or expected soon, WLS is the next best option. ME(OLS) and (OLS) will of course work the same, and all these methods should be implemented with the constraint of non-negative edge weights. Clustering methods such as ST or NJ are probably the least desirable, as they do not allow a constraint to give non-negative edge lengths. Further, because of their hierarchical clustering pattern, they can arbitrarily put more weight on certain edges in the tree.

Overall, perhaps the best reason for not relying to heavily on distance based methods to estimate edge lengths is that ML is expected to do this most efficiently, and it costs little time or effort to save a few trees and do this in PAUP*. The simulations of Kuhner and Felsenstein (1994) is consistent with this expectation. We suggest a part of the reason ML can do considerably better than distances, is that it retains information on ancestral node states, and it does this even better when many nodes are close together. Distances, loose this information. So consider in the limit an ML method which pretty much 'knew' the ancestral states. The distance to the tips is then in two parts. If you take the variances of these two half distances, it will generally be considerably lower than the variance of a single distances measured through that ancestral node, but without being able to use it as a check on the inferred extra changes (e.g. with iterated Ml distances). Additionally, PAUP* offers the constraint of a molecular-clock on ML, which should further reduce sampling variance.

A valid instance to use distances is with good evidence for a homogeneous non-reversible model, and in these cases the LogDet distance with adjustment for each sequences base composition (e.g. the Paralinear distance of Lake 1994) is expected to return distances which are clock-like in their behaviour. Another is when stationarity and reversibility are clearly incorrect and here the LogDet-Paralinear distance can provide better estimates of edge length than any of the reversible model distances and reversible ML (See Waddell 1995, section 3.4).

16.36 Editing data for use in conjunction with a distance based analysis.

This section looks at some of the ways sequence data can be edited in order that distances better fit the expectations of additivity with minimum variance and minimum systematic error. PAUP* offers a number of alternatives to which sites to include when making comparisons. One is to ignore all sites with any gaps. An alternative is to count the distance across all pairwise aligned sites ignoring only those gaps between pairs of sequences, which can give rise to the problem mentioned earlier. A further step in this direction is to infer the state at a site with unknown state (e.g. due to an unidentified base) by an iterative method. For example, if a site shows up as "AY", then the challenge is to infer what the Y actually is. Assuming equal base frequencies, plus i.i.d. and stationary evolution, and a Poison process of evolution, it is 50 / 50 the Y is "C' or "T". However, letting base frequencies be unequal, a better guess at the relative likelihoods of it being "C' or "T" is fC / (fC + fT) versus fT / (fC + fT). This is a first pass estimate. If we then take the overall frequencies of the identified pairs and the inferred pairs, we can refine the estimates of the base frequencies, and re-infer each unidentified base. Doing this over, the base frequencies converge to a stable or stationary value. The same method is used by Felsenstein (1993).

If a pair such as "CY" arises, a further complication is that the Y could be either a ”C” and so no-change or a ”T” and so inferring a transition. Clearly, now the inference of the identity of the ”Y” is also complicated by the relative frequencies of no change (estimated by the pair ”CC”) or a transition (estimated by the pair ”CT”). In this case, the initial estimates of ”CC” and ”CT” are based on fCC and fCT (assuming only an i.i.d. process). In the next round, the estimates of all the fij’s are refined and so on to an iterative solution. Clearly, there are also intermediate sets of assumptions. For example, that the process is time reversible, so fct = ftc in expectation, and an average of them is their best estimate. PAUP* presently utilizes just the most general assumptions, as it is expected the gains from matching the inference to the distance estimate will generally be small.

Another issue, often overlooked in phylogenetic analyses, is that insertions / deletions regions may be evolving distinctly from conserved regions. This seems especially likely in functional sequences, where it seems likely the regions affected by insertions and deletions may be evolving faster than average. If insertion / deletion regions are incorporated in pairwise comparisons, then they will lead to some distortion with respect to the additivity of distances. For example, two sequences may be closest relatives, but because they both share an insertion region of higher average rate may appear relatively more distant from each other than they do to less closely related sequences which do not share this insertion. This could lead to distance methods not placing them as closest relatives. In instances where this is likely, it would seem safest to estimate distances from just those sites shared amongst all sequences (and not just shared by pairs). Here, the inference method of the previous paragraph could still be used to infer bases which are deemed present, but uncertain. Removing gaps and deletions would seem safest when you are interested in resolving the deepest branches in the tree.

Another technique which is not yet automated in PAUP*, but which can be useful is to remove that portion of sites which are evolving so fast as to essentially contribute no useful information to distance estimates (irrespective of whether a distribution of site rates is allowed, see Waddell 1995, Ch. 4). It has been shown analytically, that such sites may contribute only noise, and lead to all distance estimates being unreliable (Waddell, 1995). A likely application of such an approach might be with rRNA and deep divergences, where it is clear some sites have changed so many times their either random e.g. only reflecting the overall base composition. Such rapidly evolving sites may be inferred by their retention index or an ML estimate of their rate on a likely tree (preferably the best estimate(s) at hand), and the most extreme x% of sites identified for deletion from the data, followed by reassessment of the phylogeny. This approach can be especially useful when using LogDet on sequences with highly unequal site rates (Waddell 1995, Ch. 3).

16.36.1 *Final recommendations*

(1) Edit the sequences in a sensible manner. If distances are not small and there is evidence that sites in regions with deletions may be evolving distinctly form other sites (e.g. on average at a faster rate) then careful consideration should be given to dropping sites with deletions from the data before distance estimation (PAUP* offers an option to do this during distance estimation). A similar recommendation can be made for ML analyses, although these are expected to be more robust. If there are sites in the data which are clearly very close to saturation, or at least much faster than the average sites, there can be significant gains by identifying and removing them. If sites clearly fall into groups with distinct properties (e.g. first second third positions) then editing to conform to these is desirable. In either case, a distance matrix from each set of sites can be added together legitimately, if each is expected to have evolved by the same tree topology (see Waddell, Appendix 3.4 for a proof). This is especially important for use with the LogDet, since a non-stationary model, with unequal site rates, is expected to show a distinct base composition in each site-rate class.

(2) One the edited sequences test the following factors:

(a) stationary base composition

(b) symmetric F matrices (i.e. consistent with a reversible model)

(suitable tests can be made using the non-intersecting paths methods described herein, or the more expensive GLS tests such as Rzhetsky and Nei 1995).

If (a) and (b) cannot be rejected, then the model is likely to be either reversible (or at least clock-like). The next step is then to decide on which reversible distance(s) to use. Tests of different reversible models comparing predicted and expected F matrices on distinct paths through the likely tree are given earlier. GLS tests are described in Rzhetsky and Nei (1995) while linear invariant tests of Kimura 3ST and submodels are given in Waddell (1995). Alternatively, testing the fit of models with ML could be performed (see section *). In either case, if you are quite confident a stationary model, homogeneous model is a reasonable assumption, you can reduce sampling errors further by estimating the parameters in the form of R selected, and making iterated ML estimates of all distances.

If either (a) or (b) appear false, then time reversibility of the model is rejected. In these instances, the LogDet appears to be the most robust estimator we presently have. Carefully compare its results with more specialised distances (e.g. the K2P, the GTR distance). Another alternative is to look to either increase or reduce the number of states (in either case focusing on rarer changes), in the hope that the non-stationarity is restricted to the 4 nucleotides, and / or the slower rates markedly reduce its potential impact.

(c) site rate homogeneity (i.e. do sites have identical rates).

(d) test for a distinct base composition of the constant sites.

(section * outlines a variety of ways of estimating either the number of invariant sites or the shape parameter for continuous distributions such as the G).

If (c) is rejected (i.e. site rates are not identical) then this factor should be accounted for in estimating a distance. It is useful to be able to make a refined estimate of the form of site rate variability. If a reversible model is being used, one can then use ML or a count of the parsimony changes to check whether a pinv, or a G, or a model with both fits the data best. In all cases PAUP* offers time reversible distances allowing for each option. Also, do not overlook editing as a way to better refine site rate distributions. Whenever there is evidence of site rate homogeneity, then it makes sense to test whether the conservative sites (e.g. represented by the constant sites) have an equal base composition to the variable sites (for this purpose a standard contingency table test can be used to give a conservative result, Waddell 1995). If this test (d) is significant, PAUP* offers an option to modify F in proportion to the constant site base frequencies.

In if either (a) and / or (b) is correct and either (c) or (d) is correct, then the LogDet should be considered with modification to allow for a fraction of invariant sites. This is not ideal, and hopefully in future an option of editing sites into rate categories, applying the LogDet to each, then summing the resultant distance matrices will be implemented (Waddell, 1995). Additionally, because the LogDet often has quite reasonable variance in relation to the other distances, and because in our experience of real analyses it is a shift from a reversible to a LogDet distance that often sees the greatest difference in the overall tree, we recommend it be considered routinely and contrasted with reversible distances unless sequences are particularly closely related.

In terms of tree selection, the most efficient methods with sufficient data (e.g. GLS have yet to be implemented). If WLS is available, we recommend its use with weights being the inverse of the variance estimated by the delta method formulae given earlier (or with a similar approximation if iterated ML distances are used). Studies so far also indicate that the constraint that all edges be non-negative can make a substantial improvement (Kuhner and Felsenstein, 1994), and hopefully this is achieved with a constrained optimisation. It is out expectation that methods such as FM, ME (OLS) and OLS will often perform similarly (and not quite as well as WLS) when edges are constrained to be non-negative. There is, however, no harm in considering the allowance of negative edges and reporting the differences seen.

Clustering algorithms such as ST and NJ appear to work quite reasonably. For closely related sequences there are good indications that parsimony methods are superior (e.g. Charleston 1994, Charleston et al. 1994, Waddell et al. 1997), but for deeper divergences, we see these as valuable methods, especially since they can run very quickly and easily allow bootstrapping. In a very real sense, a combination of NJ, bootstrapping, a variety of distances and quick tests of features of the model, plus allowance for unequal rates across sites, are an ideal data exploration technique. In this way the possibilities of the data can be explored, and the more intensive methods (either ML or distances) used in the range of values yielding results of interest.

Lastly, be aware that sequences may be evolving by a model which is quite unlike a simple i.i.d. Markov model, even one as general as that the LogDet is based on. Possible reasons for this include changing molecule function (Lockhart et al. 1996), the heterogeneity of process all along functional molecules, or second order Markov evolution mediated by stabilizing selection (Waddell et al. 1996). In these cases, it is well worth visualising the tree-like structure in the data using a technique such as Splits Tree (Bandelt and Dress 1992) or the distance Hadamard (Hendy and Penny 1993). If it does not appear reasonable, realignment, editing, alternative state assignments or distance estimators may all be tried to improve a demanding area of phylogenetic analysis.

It was on the note of data exploration techniques that this chapter began, and it now finishes on this note. Distances transformations offer a far wider range of realistic models than are presently implemented for ML. They are fast to very fast methods, which all both in depth tree searching and rapid bootstrapping. Their major weakness appears to be one that shows up in direct contrast to ML, which is probably often, but not always more robust and statistically efficient. They offer a series of methods that allow interactive run times on moderate to large data sets (e.g. 20 to 100 taxa), so that factors such as editing, inclusion and exclusion of certain taxa, model assumptions, etc. can be evaluated by the biologist. These are qualities which if ignored deplete the range any phylogenetic analysis, and to which Tukey alludes in the opening sections quote on exploratory methods.

References in second file.

Cut bits and scraps

Talk about technique based methods, e.g. Gifi (1989). Basically you have at your disposal a set of techniques known to work well. Basically we don’t know this for sure, but knowing a bit about the real process of evolution, and a bit about the sensitivity of techniques in the face of models with these features we can suggest some general guides to which technique to apply (note this approach is distinct and complimentary to attempting to fit a model to the data. The later approach tends to extract more from the data, and be more intensive, the results of such studies, if general enough, can usefully feed back into background on techniques).

(might I illustrate these sections with an example data file? i.e. mention something, then say you might like to try this on this data set).

There are now a bewildering array of distances to use, and many of them will give the same tree with very similar bootstrap support, despite being based associated with models which may have very different likelihood scores. Conversely there will be times when two distances both initially seem appropriate, but will give trees with distinct bootstrap support. These paradoxes are encountered more frequently with the more ancient, and often more interesting, divergences. In these situations is quite certain that evolution is not i.i.d., and none of the models may be expected to uniformly give a good approximation to the evolutionary process. This posses the dilemma of what phylogenetic information can be retrieved and relied upon. An alternative use of distance methods is to infer edge lengths on trees (weighted trees). This is especially useful in inferring divergence times of species, and the relative rates of mutation in different lineages. This use of distances is slightly different to the first and is considered second.

At present, it seems most reasonable to still aim clearly for a distance transformation which takes into account all the major aspects of the model. The tests described in section * above, allow identification of when base composition is clearly fluctuating between species. On the assumption that if it is marked enough effect to be clearly detectable, and that it could pose a potential problem for tree selection, then sensible alternative is to use a distance which is robust to this factor. The best available at present is the LogDet. If in the same example there is also evidence for unequal site rates, and estimate of the proportion of invariant sites should be made, then used to modify the LogDet to a CSR-LogDet transform in order to maximise its robustness. In reality, many functional sequences diverged for more than 200 million years will show both these factors clearly. Concern about the impact of the increased variance of the LogDet can be addressed by a comparison of the bootstrap support obtained with the LogDet in comparison to a low variance estimator such as the K2P distance. If the bootstrap support with the LogDet is close to K2P distance in many parts of the tree, then where they differ substantially should be called to attention and is unlikely to reflect just the increased variance of the LogDet (Waddell, 1995). Using distances to explore when different models infer substantially different trees.

Here we suggest an extension of the common wisdom that if two or more distinct methods give similar results there may be grounds to have confidence in their conciliance, and where they differ alerts us to important characteristics of their evolution. This can be considered an extension of the grass roots philosophy that if unweighted parsimony and Jukes Cantor - neighbor joining both give the same tree, it is probably right. While this perspective has now been shown to often be too simplistic with more ancient divergences (e.g. Lockhart et al. 1994, 1996) and in simulations (Huelsenbeck ?) it can be extended and made more rigorous.

If all sequences are very similar (e.g. no two show an observed distance of more than 0.05) then most all of the transformations will make little difference to the relative sizes of the observed distances. In this type of situation we would expect observed distances, time reversible distances, LogDet, and with or without compensation for unequal rates, to result in very similar trees. The differences between trees will most often be due to sampling error, which will tend to be exaggerated with the distances based on more general models. Unless the more general distances appear to be indicating quite different support for a partition in the tree, then it seems likely this explanation is adequate for the data at hand, and the use of a simple or no distance correction is likely to yield a reliable estimate of the tree with lower sampling error than the more general distances.

As distances become larger, aspects of the evolutionary process can begin to exert an appreciable influence on the relative sizes of transformed distances. A good example of this is mammalian mtDNA. The highly unequal transition to transversion rate (often over 20:1) can begin to exert appreciable differences in the relative size of distances which do (e.g. Kimura 2ST, HKY) versus those which don't (Jukes Cantor and Tajima and Nei respectively) allow for this. Base compositions are often highly unequal, for example rare in G, but at least for closely related species often do not deviate substantially in base composition. This factor can see transformations which take this feature into account (e.g. TN, HKY) result in considerably more non-linear transformations than those which do not (e.g. JC and K2 respectively). Contiguous regions of mtDNA also tend to show very unequal site substitution rates (e.g. Waddell and Penny 1996) and its not uncommon to expect more than 50% of sites to be invariant, or to infer a G distribution with a shape parameter of less than 0.5 (hyper-exponential in shape). Consequently the use of a distance which takes this factor into account, versus one that does not, can be considerable in terms of relative size of inferred distances. All of these factors (plus unequal rates of the two transition types) tend to act in a non-linear, super cumulative way. Consequently, with mammalian mtDNA is often a good idea to take particular note of how application of the general time reversible distance with a suitable distribution of rates across sites may change the inferred tree and its bootstrap support from simpler submodels.

With the previous examples we still tend to be in the region where we can expect a reasonable match of gross features of the evolutionary process to the assumptions of distances. However, even here some factors can become troublesome and are perhaps best dealt with by data editing (followed by checking for changing bootstrap support). A good example are transitional changes. As these approach saturation, they will tend to be less discriminatory in terms of relative distances and more prone to systematic errors (e.g. a preponderance for some sites to show only transitional changes). Similarly, past some tens of millions of years third positions may be suspected of having saturated. Reliance upon just transversion distances or recoding to 2 states will tend to alleviate both these problems (since transitions tend to occur most frequently at third positions). It is interesting to then compare the bootstrap support for the tree under each alternative, to see what difference it has made.

Going further back in time again, our mtDNA sequences may begin to show more aberrant behaviour. Deep within the mammals, base composition can vary between taxa, and this suggests particular attention should be paid to CSR-LogDet transformations and the trees they yield. The effect of the exclusion of third positions would be interesting to evaluate, as these sites may be contributing little reliable information by this time. Alternatively, estimating just transversions should hopefully also offer robustness to shifting base compositions and trees from these distances will hopefully agree with the CSR-LogDet tree. If they do not, this is an interesting point that deserves to be highlighted in the analysis.

Further back in time again we should be questioning the assumptions that all our distance transformations rest upon. Under any i.i.d. model, even with selective constraints of different strengths on each site, it is not expected that substitutions (even at sites with strong constraints) will persist any longer than neutral changes. Thus the fixation of "informative" changes for periods of hundreds of millions of years are not explained by any of our models, and need to be considered some form of covarion model (i.e. a site which has undergone a neutral substitution has then experienced stabilization of that state by the imposition of a selective disadvantage on any other state). Such models while very interesting, may pose serious problems to our current i.i.d. methods. In these situations, we suggest that the results of CSR-LogDet transforms should be compared and contrasted with the results of the less general distances, and especially distances based on transversional or transitional changes only. Such differences should be noted clearly in the analysis, and if possible explained. In this situation data editing, especially by the removal of sites which apparently evolve fastest, should be considered in helping to clarify which partitions seem most likely to be correct.

Guide lines for data editing.

Using distance transformations to give more accurate edge length estimates Summary References: Cut bits, do read if anything in here catches your fancy for inclusion in the chapter. David, I had though that it might be good to have a history of the methods, because there are so many miscitations and independent developments. Now I'm thinking more it can be done more subtly in the text. Appendix ? History of tree estimation from pairwise distances Appendix ? History of development of distance transformations for DNA sequences Cut bits: Adhoc modifications to distances to compensate for non-stationary base compositions – NOT THIS VERSION References: Kishino, H., and M. Hasegawa. (1990). Converting distance to time: An application to human evolution. Methods in Enzymology 183: 550-570.

Possibly to work in

Simulations and analytic calculations with Equation 2 suggest it yields an ML estimate of the true distance given just and a distribution of rates across sites. By ML estimate, we mean the distance which will minimise the G2 statistic between F and when all entries in R (i.e. both components of R = PS) are simultaneously optimised. ML estimators often have the desirable property that as c becomes large, they have the minimum possible sampling variance of all estimators for that model. Consistent with this, under models known to have ML distance estimators (e.g. those of Jukes and Cantor 1969, Kimura 1980 and 1981, see Saitou 1990) equation 5 returns identical variance estimates to the delta method variances of these estimators. In our simulations and bootstrap analyses, equation 2 often has less variance than distance estimators which are known not to be ML estimators. These include the 3P distance of Tamura (1992) and the six parameter distance of Tamura and Nei (1993) (see Zharkikh 1994). In this sense it is often a better distance to use for estimating evolutionary trees than these other estimators, especially when distances become larger and / or base composition unequal (but stationary).

References: Farris, J. S. 1972. Estimating phylogenetic trees from distance

matrices. Am. Nat. 106:645-668. Sattath, S., and A. Tversky. (1977). Additive similarity trees. Psychometrika 42: 319-345. David and Paul. I have been thinking about tree inference. What I wonder might occur in samples is that there are multiple semi-independent predictors of what the true tree might be. These include the site patterns, the general form of the distance matrix, and the sum of edge lengths on the reconstructed tree. It seems reasonable that these three predictors will be somewhat independent of one another in many applications, so that if error makes the distance matrix distorted enough to seriously mislead NJ (for example) it might not do the same for site patterns. I think in our simulations, we should check this factor out, and see if a conciliance of methods can help in estimating the tree (in situations where the data comes from an i.i.d. model which is being assumed). This is an extension of J-Y Kim's results.

It seems strange, but the field of distance based phylogenetics, perhaps more than any other, has seen repeated independent descriptions and developments of methods. This repeated derivation of similar and equivalent formulas, apparently without knowledge of what others have done, is also prevalent in distance based transformations (e.g. five or more published derivations of the basic LogDet method, see below). Factors contributing to this, include statisticians considering the problems but not publishing in mainstream phylogenetic journals, derivation of methods without evaluating their use, and a previous general rarity of distance based methods available in computer software. We hope this work helps to better integrate the field.

If there is anything in here David or Paul, that you want say so:

The variances of time reversible distances for a set of homologous sequences can be further improved by considering information from the set of sequences as a whole. Rearranging Equations (1) and (2) we have,

F(exp) = Pexp(Rdij) = Pexp(PSdij) (Equation 4) or F(exp) = PM-1(Rdij) = PM-1(PSdij) (Equation 5) respectively, where F(exp) is that predicted by the process. Here R is again a normalised rate matrix with off diagonal entries summing to one (and P is the equilibrium value for R). Given an accurate estimate of R for all sequences in the tree, then the variance of d (under the time reversible model) can be further reduced by fixing R (and hence P) when making each distance estimate, then altering dij to minimise lnL = . A statistically efficient way to obtain R for a set of more than 2 sequences is by sequence ML (Swofford et al., 1996)(Felsenstein 1981, Adachi and Hasegawa 1994). This basic approach is analogous to Felsenstein (1993) making ML estimates under the Kimura 2ST and HKY models (program DNADIST and documentation, except in the latter case he uses just unweighted averages to estimate P).

For two or more sequences, this can be solved by counting each constant aligned site just once, but each site with a substitution gives two counts. For more than two sequences, the problem becomes more protracted, and if not taken into account, will lead to rejecting the null hypothesis too often (i.e. the test will suggest base composition of A not equal to G, too often). (Note, that with two identical sequences being compared, the bias to the test can immediately be very high). A conservative test would be to take the average base composition over all sequences, but multiply these by just c (the number of sites) before making the tests. An improvement would be to count the sequence length as the number of constant sites, plus Sxini (where xi is the number of bases in an aligned column of sites and ni is the frequency of such columns, for i = 2 to 4). Having made this adjustment, an insignificant test result might be taken as a sign that it is best to consider a distance where these entries are not distinct in p.

Testing whether relative rates of change in S are distinguishable, can be accomplished using a similar method. A transition matrix P (for a pair of sequences) = ln(R) and both matrices have corresponding entries in equivalent positions (e.g. Zharkikh 1994, Lewis and Swofford 1996). Since F = PP, F can be considered to be PXP, where X will be symetric and have a pattern of entries the same as S. We could then aim to test which entiries in X are equivalant. Again, we also run into problems if we attempt to estimate X from more than a pair of sequences (due to correlated site information).

Another Way to estimate F is using parsimony to estimate inferred changes along all edges in a best estimate tree (Dayhoff et al. 1978). For our purpose, this method is preferable, since it much reduces the problem of the same change being counted multiple times (i.e. correlations). Each change inferred by parsimony is like one more roll of the dice, giving one of the six types of change shown in table * (there are up to 12 types of change identifiable, but without the root, only six are groupable, as is appropriate for a time reversible model). If parsimony suggests multiple changes, it may be reasonable to divide them up equally. However, a weighting like that used by Rogers and Swofford (1996) to get initial edge lengths for an ML optimization (modified to infer ancestral states) should work better if edges were long and unequal in length (an alternative method might be most parsimonious likelihood to infer ancestral states, Rodrigo ref.). Comparing the resultant six estimates of sac, sag, sat, scg, sct and sgt, is done with the same types of contingency table test applied to p. That is: Firstly, can we reject all entries being equal. If not all non-diagonal entries in S may be set as equal. If so, find that pair of patterns allowed to be unequal that make the most X2, and set them as different. Continue until the test is no longer significant or all six categories have been implied distinct.

Making test on P and S separately is fine, unless entries in them are non-independent. In this case (applicable also when there is independence) it is best to test once the observed pooled estimate of F, against its expectation under the model, as described earlier. An estimate of R may be made by considering P and S separately, but is concluded with a test of Fpooled(obs) and Fpooled(expt). This test, like any contingency table test, can be made with the X^2 or G2 statistic, and as long as all cells are >5, a comparison of the test statistic to a c2 distribution is reasonable. If there is already confidence that Fpooled(obs) is symmetric, the cell F_{ij} and F_{ji} can be grouped to increase power. If the test is not significant, one can look to reduce the number of parameters assumed in R, and continue searching for more restrictive models which are not rejected (and which don't increase the test statistic by 2 or more units).

A worked example with mtDNA sequences. David S, please generate me a parsimony F matrix for the mtDNA data, and a linear sum of F matrices. Testing for equaility of base compostion.

Such a test is implemented in PAUP* assuming each sequence to be an independent sample of base composition. This is not correct (sequences will usually show highly correlated base compositions due to shared common history). Effectively this results in rejecting the null hypothesis (of stationarity) too often (i.e. the test is liberal). Rzhetsky and Nei (1995) have proposed a GLS based test to avoid this problem.

Modifications to reduce this conservativeness would be to treat c as the total number of changes (including singletons) reconstructed (or implied) on the true tree (or in practice its best estimate). This could be done with parsimony or any other method (although it would seem appropriate to limit it to a maximum of t changes per site). This works because every time there is a change, its like rolling a dice to see which base comes up. Neither of these tests is exact, but they may serve as a good guide. All these tests should consider the effect of invariant sites and possibly their distinct base composition form the variable sites (see Waddell and Steel 1996). Unequal rates across sites, can also complicate these tests.

Since often the researcher will have no prior expectations of base composition, a contingency table test of all bases being equal may be conducted first. This could be followed by successive relaxations of equal base composition until this type of test is no longer significant (or all four bases are recognised as being at different base compositions). When bases are recognized as being at unequal frequencies, these modifications can be set into the custom distances in PAUP* by setting the appropriate boxes to specify P.

Combined with invariant sites models, or classification of sites into rate classes, the LogDet can be expected to return near additive distances when rates across sites vary. In practice the robustness of the LogDet transform to non-stationary base compositions can be seen to yield more biologically realistic trees in studies in Lockhart et al. (1994) and Waddell and Lockhart (submitted) when removing specific proportions of invariant sites.

As a simple example, consider the human to siamang transversion distance for the data of Horai et al. 1996. If we assume site rates are either i.r. (or follow a G distribution, a = 0.25) then the equi-frequency distance 2-state model distance (e.g. the transversions only of the K 2ST model) is .. (), the 2-state unequal base frequency (e.g. the HKY model) is (), while under the GTR model it is the sum of the transversion entries in PRt, that is (). These transversion distances are not large, so the differences are fairly minor.

Gascuel, O. (1994). A note on Sattath and Tversky’s, Saitou and Nei’s, and Studier and Keppler’s algorithms for inferring phylogenies from evolutionary distances. Mol. Biol. Evol. 11: 961-963.

David, you were right, FM does use 1/Dij, where Dij is the data matrix distance. Below is what I had written earlier. I have rewritten it above to correct my mistake. I suspect that an iterated WLS could work very well. It may be slower than WLS, but it should still be quite fast, and might make a good criterion for a final ’sweep’ in the neighborhood of the best trees. I leave it once for you to consider for future versions, but cut it now?

The FM criterion is both simpler and more sophisticated than these applications of WLS. It is simpler in that it does not calculate the variance explicitly, rather it assumes that it increases as the square of the distance (i.e. distance doubles then variance increases four times), and the distances used are those on the latest inferred tree. Thus, in the FM method wx = 1/(dTx)2. Stated another way, the standard deviation of a distance is proportional to the length of a path through the tree being evaluated. This expectation has been little explored. Later we show that in some cases it can be quite realistic, whereas in others it may not be very accurate.

The FM criterion is sophisticated because, like the iterated GLS method, it does update the variance according to the tree being calculated (i.e. it uses the expected distance). This is potentially important, since the commonly used X^2 criterion is S(observed - expected)2 / expected, and while the statistic S(observed - expected)2/observed is closely related, it is known to have decreased statistical power. Put another way, when there is an unexpected deviation from the model, it is worse if the big distance is seen in place of the small distance, than if the small distance is seen in the place of the big distance (i.e. all other things are equal except the big expected distance has the bigger variance, making the deviation seem more likely).

As we show later, neither assumption exactly reflects what is happening with DNA sequences, although the FM method uses a near optimal weighting for many distances assuming an underlying G distribution when the shape parameter is small (less than 1). Generally, it is desirable to set wx to be the inverse of the variance expected for a distance, as this can now be easily calculated for all the homogeneous rate models.

Bibliography

Adams, I., E. N. 1972. Consensus techniques and the comparison of taxonomic trees. Systematic Zoology 21:390–317.

Adams, I., E. N. 1986. N-trees as nestings: complexity, similarity, and consensus. Journal of Classification 3:299–317.

Agressti, A. 1990. Categorical Data Analysis. Wiley, New York.

Akaike, H. 1974. A new look at the statistical model identification. IEEE. Trans. Autom. Contr. 19:716–723.

Archie, J. W. 1989. Homoplasy excess ratios: new indeces for measuring levels of homoplasy in phylogenetic systematics and a critique of the consistancy index. Systematic Zoology 38:253–269.

Bandelt, H. J. and A. W. M. Dress. 1992. Split decomposition: A new and useful approach to phylogenetic analysis of sequence data. Molecular Phylogenetics and Evolution 1:242–252.

Barry, D. and J. A. Harrington. 1987. Asynchronous distance between homologous dna sequences. Biometrics 43:261–276.

Bishop, M. J. and A. E. Friday. 1985. Evolutionary trees from nucleic acid and protein sequences. Proceedings of the Royal Society of London, B 226:271–302.

Bremer, K. 1990. Combinable component consensus. Cladistics 6:369–372.

Brent, R. P. 1973. Algorithms for Minimization Without Derivatives. Prentice-Hall, Englewood Cliffs, NJ.

Bulmer, M. 1991. Use of the method for generalized least squares in reconstructing phylogenies from sequence data. Molecular Biology and Evolution 8:868–883.

Bulmer, M. 1994. Theoretical Evolutionary Ecology. Sinauer Associates, Sunderland, Massachusetts.

Buneman, P. 1971. The recovery of trees from measures of dissimilarity. Pages 387–395 *in* Mathematics in the Archaeological and Historical Sciences (F. R. Hodson, D. G. Kendall, and P. Tautu, eds.) 1st ed. Edinburgh University Press, Edinburgh.

Camin, J. H. and R. R. Sokal. 1965. A method for deducing branching sequences in phylogeny. Evolution 19:311–326.

Cavalli-Sforza, L. L. and A. W. F. Edwards. 1967. Phylogenetic analysis: models and estimation procedures. Evolution 32:550–570.

Chang, J. T. 1996. Inconsistency of evolutionary tree topology reconstruction methods when substitution rates vary across characters. Mathematical Biosciences 134:189–216.

Churchill, G. A., A. von Haeseler, and W. C. Navidi. 1992. Sample size for a phylogenetic inference. Molecular Biology and Evolution 9:753–769.

Constantinescu, M. and D. Sankoff. 1986. Tree enumeration modulo a consensus. Journal of Classification 3:349–356.

Cox, D. R. and D. V. Hinkley. 1974. Theoritical statistics. Chapman and Hall.

DeBry, R. W. and N. A. Slade. 1985. Cladistic analysis of restriction endonuclease cleavage maps within a maximum-likelihood framework. Syatematic Zoology 34:21–34.

Donoghue, M. J. and W. P. Maddison. 1986. Polarity assessment in phylogenetic systematics: a response to meacham. Taxon 35:534–545.

Edwards, A. W. F. 1972. Likelihood. Cambridge University Press.

Edwards, A. W. F. and L. L. Cavalli-Sforza. 1964. Reconstruction of evolutionary trees. Pages 67–76 *in* Phenetic and Phylogenetic Classification (H. V. H. McNeill J., ed.) 1st ed. Systematics Association Publication no. 6, London.

Edwards, E. J., C. P. Osborne, C. A. E. Stromberg, S. A. Smith, and G. Consortium. 2010. The origins of c4 grasslands: integrating evolutionary and ecosystem science. Science 328:587–591.

Efron, B. 1979. Bootstrap methods: Another look at the jackknife. Annals of Statistics 7:1–26.

Faith, D. P. 1991. Cladistic permutation tests for monophyly and nonmonophyly. Systematic Zoology 40:366–375.

Farris, J. S. 1969. A successive approximation approach to character weighting. Systematic Zoology 18:374–385.

Farris, J. S. 1970. Methods for computing wagner trees. Systematic Zoology 19:83–92.

Farris, J. S. 1972. Estimating phylogenetic trees from distance matrices. American Naturalist 106:645–668.

Farris, J. S. 1977. Phylogenetic analysis under dollo's law. Systematic Zoology 26:77–88.

Farris, J. S. 1982. Outgroups and parsimony. Systematic Zoology 31:328–334.

Farris, J. S. 1988. Hennig86, version1.5. distributed by author. Tech. rep. Port Jefferson, N. Y.

Farris, J. S. 1989a. The retention index and homoplasy excess. Systematic Zoology 38:406–407.

Farris, J. S. 1989b. The retention index and the rescaled consistancy index. Cladistics 5:417–419.

Farris, J. S. 1991. The Complete Cladist. Universityy of Kansas Museum of Natural History, Special Publication Number 19, Lawrence, Kansas.

Farris, J. S., V. A. Albert, M. Kallersjo, D. L. Lipscomb, and A. G. Kluge. 1996. Parsimony jackknifing outperforms neighbor-joining. Cladistics 12:99–124.

Farris, J. S., M. Källersjö, A. G. Kluge, and C. Bult. 1995. Testing significance of incongruence. Cladistics 10:315–319.

Felsenstein, J. 1973. Maximum-likelihood estimation of evolutionary trees from continuous characters. American Journal of Human Genetics 25:471–492.

Felsenstein, J. 1978a. Cases in which parsimony and compatibility methods will be positively misleading. Systematic Zoology 27:401–410.

Felsenstein, J. 1978b. The number of evolutionary trees. Systematic Zoology 27:27–33.

Felsenstein, J. 1981a. Evolutionary trees from dna sequences: A maximum likelihood approach. Journal of Molecular Evolution 17:368–376.

Felsenstein, J. 1981b. A likelihood approach to character weighting and what it tells us about parsimony and compatibility. Biologica Journal of the Linnaean Society 16:183–196.

Felsenstein, J. 1983. Parsimony in systematics: biological and statistical issues. Annual Review of Ecology and Systematics 14:313–333.

Felsenstein, J. 1984. Distance methods for inferring phylogenies: A justification. Evolution 38:16–24.

Felsenstein, J. 1985. Confidence limits on phylogenies: An approach using the bootstrap. Evolution 39:783–791.

Felsenstein, J. 1986. Distance methods: A reply to farris. Cladistics 2:130–143.

Felsenstein, J. 1987. Estimation of hominoid phylogeny from a dna hybridization data set. Journal of Molecular Evolution 26:123–131.

Felsenstein, J. 1988. Phylogenies from molecular sequences: inference and reliability. Annual Reviews of Genetics 22:521–565.

Felsenstein, J. 1991. Phylip (phylogeny inference package), version 3.4. distributed by the author. Tech. rep. University of Washington, Seattle, Washington.

Felsenstein, J. 1992. Phylogenies from restriction sites: a maximum-likelihood approach. Evolution 46:159–173.

Felsenstein, J. and G. Churchill. 1996. A hidden markov model approach to variation among sites in rate of evolution. Molecular Biology and Evolution 13:93–104.

Fisher, R. A. 1922. On the mathematical foundations of theoretical statistics. Philosophical Transactions of the Royal Society, A 222:309–368.

Fishman, G. S. and L. R. Moore. 1982. A statistical evaluation of multiplicative congruential random number generators with modulus 231-1. Journal of the American Statistical Association 77:129–136.

Fitch, W. M. 1971. Toward defining the course of evolution: minimal change for a specific tree toplogy. Systematic Zoology 20:406–416.

Fitch, W. M. 1981. A non-sequencial method for constructing trees and hierarchical classifications. Journal of Molecular Evolution 18:30–37.

Fitch, W. M. and E. Margoliash. 1967. Costruction of phylogenetic trees. Science 155:279–284.

Fukami, K. and Y. Tateno. 1989. On the maximum likelihood method for estimating molecular trees: Uniqueness of the likelihood point. Journal of Molecular Evolution 28:460–464.

Funk, V. A. and D. R. Brooks. 1990. Phylogenetic Systematics as the Basis for Comparative Biology. Smithsonian Institution Press: Washington, D. C.

Gascuel, O. 1997. Bionj: an improved version of the nj algorithm based on a simple model of sequence data. Molecular Biology and Evolution 14:685–695.

Gaut, B. S. and P. O. Lewis. 1995. Success of maximum likelihood phylogeny inference in the four-taxon case. Molecular Biology and Evolution 12:152–162.

Goddard, W., E. Kubicka, G. Kubicki, and F. R. McMorris. 1994. The agreement metric for labeled binary trees. Mathmatical Biosciences 123:215–226.

Goldman, N. 1990. Maximum likelihood inference of phylogenetic trees, with special reference to a poisson process model of dna substitution and to parsimony analyses. Systematic Zoology 39:345–361.

Goldman, N. 1993. Statistical tests of models of dna substitution. Journal of Molecular Evolution 36:182–198.

Goldman, N., J. P. Anderson, and A. G. Rodrigo. 2000. Likelihood-based tests of topologies in phylogenetics. Systematic Biology 49:652–670.

Goldman, N. and Z. Yang. 1994. A codon-based model of nucleotide substitution for protein-coding dna sequences. Molecular Biology and Evolution 11:725–736.

Goloboff, P. 1993. Nona. version 2.0. Tech. rep. Fundaci e Instituto Miguel Lillo.

Gould, R. 1988. Graph Theory. Benjamin-Cummings: Menlo Park, California.

Harary, F. 1969. Graph Theory. Addison-Wesley: Reading, Massachusetts.

Hasegawa, M. and M. Fujiwara. 1993. Relative efficiencies of the maximum likelihood, maximum parsimony, and neighbor-joining methods for estimating protein phylogeny. Molecular Phylogenetics and Evolution 2:1–5.

Hasegawa, M. and H. Kishino. 1989. Confidence limits on the maximum-likelihood estimate of thehominoid tree from mitochondrial- dna sequences. Evolution 43:672–677.

Hasegawa, M., H. Kishino, and T. Yano. 1985. Dating the human-ape split by a molecular clock of mitochondrial dna. Journal of Molecular Evolution 22:160–174.

Hendy, M. D. and D. Penny. 1982. Branch and bound algorithms to determine minimal evolutionary trees. Mathmatical Biosciences 59:277–290.

Hendy, M. D. and D. Penny. 1984. Cladograms should be called trees. Systematic Zoology 33:245–247.

Hendy, M. D. and D. Penny. 1989. A framework for the quantitative study of evolutionary trees. Systematic Zoology 38:297–309.

Hendy, M. D. and D. Penny. 1993. Spectral analysis of phylogenetic data. Journal of Classification 10:5–24.

Hendy, M. D., M. A. Steel, D. Penny, and I. M. Henderson. 1988. Families of trees and consensus. Pages 355–362 *in* Classification and Related Methods of Data Analysis (H. H. Bock, ed.) 1st ed. Sinauer Associates, Sunderland, Massachusetts.

Hennig, W. 1966. Phylogenetic Systematics. University of Illinois Press: Urbana, Illinois.

Hillis, D. M. 1987. Molecular versus morphological approaches to systematics. Annaul Review of Ecology and Systematics 18:23–42.

Hillis, D. M. 1991. Discriminating between phylogenetic signal and random noise in dna sequences. Pages 278–294 *in* Phylogenetic Analysis of DNA sequences (M. M. Miyamoto and J. Cracraft, eds.) 1st ed. Oxford Universit Press: New York, N. Y.

Hillis, D. M. and J. P. Huelsenbeck. 1992. Sigal, noise, and reliability in molecular phylogenetic analyses. Journal of Heredity 83:189–195.

Huelsenbeck, J. P. 1991. Tree-length distribution skewness: an indicator of phylogenetic information. Systematic Zoology 40:257–270.

Huelsenbeck, J. P. 1995a. The performance of phylogenetic methods in simulation. Systematic Biology 44:17–48.

Huelsenbeck, J. P. 1995b. The robustness of two phylogenetic methods: Four-taxon simulations reveal a slight superiority of maximum likelihood over neighbor joining. Molecular Biology and Evolution 12:843–849.

Jukes, T. H. and C. R. Cantor. 1969. Evolution of protein molecules. Pages 21–132 *in* Mammalian Protein Metabolism (H. N. Munro, ed.) 1st ed. Academic Press, New York.

Källersjö, M., J. S. Farris, A. G. Kluge, and C. Bult. 1992. Skewness and permutation. Cladistics 8:275–287.

Kimura, M. 1980. A simple method for estimating evolutionary rate of base substitution through comparative studies of nucleotide sequences. Journal of Molecular Evolution 16:111–120.

Kimura, M. 1981. Estimation of evolutionary distances between homologous nucleotide sequences. Proceedings of the National Academy of Sciences, USA 78:454–458.

Kishino, H. and M. Hasegawa. 1989. Evaluation of the maximum likelihood estimate of the evolutionary tree topologies from dna sequence data, and the branching order in hominoidea. Journal of Molecular Evolution 29:170–179.

Kishino, H., T. Miyata, and M. Hasegawa. 1990. Maximum likelihood inference of protein phylogeny and the origin of chloroplasts. Journal of Molecular Evolution 30:151–160.

Kluge, A. G. and J. S. Farris. 1969. Quantitative phyletics and the evolution of anurans. Systematic Zoology 18:1–32.

Kuhner, M. and J. Felsenstein. 1994. A simulation comparison of phylogeny algorithms under equal and unequal evolutionary rates. Molecular Biology and Evolution 11:459–468.

Lanave, C., G. Preparata, C. Saccone, and G. Serio. 1984. A new method for calculating evolutionary substitution rates. Molecular Biology and Evolution 20:86–93.

Lewis, P. O. 1998. Maximum likelihood as an alternative to parsimony for inferring phylogeny using nucleotide sequence data. Pages 132–163 *in* Molecular Systematics of Plants II: DNA sequencing (D. E. Soltis, P. S. Soltis, and J. J. Doyle, eds.) 1st ed. Kluwer Academic Publishers.

Lewis, P. O. 2001. A likelihood approach to estimating phylogeny from discrete morphological character data. Systematic Biology 50:913–925.

Lundberg, J. G. 1972. Wagner networks and ancestors. Systematic Zoology 21:398–413.

Maddison, D. R. 1991. The discovery and importance of multiple islands of most parsimonious trees. Systematic Zoology 40:315–328.

Maddison, D. R., D. L. Swofford, and W. P. Maddison. 1997. Nexus: An extensible file format for systematic information. Systematic Biology 46:590–621.

Maddison, W. P. 1989. Reconstructing character evolution on polytomous cladograms. Cladistics 5:365–377.

Maddison, W. P. 1993. Missing data versus missing characters in phylogenetic analysis. Systematic Biology 42:576–581.

Maddison, W. P., M. J. Donoghue, and D. R. Maddison. 1984. Outgroup analysis and parsimony. Systematic Zoology 33:83–103.

Maddison, W. P. and D. R. Maddison. 1992. MacClade: Analysis of phylogeny and character evolution, version 3.0. Sinauer Associates, Sunderland, Massachusetts.

Maddison, W. P. and D. R. Maddison. 2012. Mesquite: a modular system for evolutionary analysis. Version 2.75. http://mesquiteproject.org.

Margush, T. and F. R. McMorris. 1981. Consensus n-trees. Bulletin of Mathematical Biology 43:239–244.

Masuda, R., J. V. Lopez, J. Pecon-Slattery, N. Yuhki, and S. J. O'Brien. 1996. Molecular phylogeny of mitochondrial cytochrome b and 12s rrna sequences in the felidae: ocelot and domestic cat lineages. Molecular Phylogenetics and Evolution 6:351–365.

McMahon, M. M. and M. J. Sanderson. 2006. Phylogenetic supermatrix analysis of genbank aequences from 2228 papilionoid legumes. Systematic Biology 55:818–836.

Meacham, C. A. 1984. The role of hypothesized direction of character in the estimation of evolutionary history. Taxon 33:26–38.

Meacham, C. A. 1986. More about directed characters: a reply to donoghue and maddison. Taxon 35:538–540.

Miyamoto, M. M. 1985. Consensus cladograms and general classifications. Cladistics 1:186–189.

Muse, S. V. 1995. Evolutionary analyses of dna sequences subject to constraints on secondary structure. Genetics 139:1429–1439.

Muse, S. V. and B. S. Gaut. 1994. A likelihood approach for comparing synonymous and nonsynonymous nucleotide substitution rates, with application to the chloroplast genome. Molecular Biology and Evolution 11:715–724.

Muse, S. V. and B. S. Weir. 1992. Testing for equality of evolutionary rates. Genetics 132:269–276.

Nei, M. and L. Jin. 1989. Variances of the average number of nucleotide substitutions within and between populations. Molecular Biology and Evolution 6:290–300.

Nelson, G. and N. I. Platnick. 1981. Systematics and biogeography: Cladistics and vicariance. Columbia University Press, New York.

Neyman, J. 1997. Molecular studies of evolution: a source of novel statistical problems. Pages 1–27 *in* Statistical Decision Theory and Related Topics (S. S. Gupta and J. Yackel, eds.) 1st ed. Academic Press, New York.

Nixon, K. C. 1999. The parsimony ratchet, a new method for rapid parsimony analysis. Cladistics 15:407–414.

Page, R. D. M. 1989. Comments on component-compatibility in historical biogeography. Cladistics 5:167–82.

Page, R. D. M. 1993. Component: Tree comparison software for microsoft windows, version 2.0. Tech. rep. The Natural History Museum, London.

Pagel, M. 1994. Detecting correlated evolution on phylogenies: a general method for the comparative analysis of discrete characters. Proceedings of the Royal Society of London, B 255:37–45.

Penny, D. 1982. Towards a basis for classification: The incompleteness of distance measures, incompatibility analysis and phenetic classification. Journal of Theoritical Biology 96:129–142.

Penny, D. and M. D. Hendy. 1985. The use of tree comparison metrics. Systematic Zoology 34:75–82.

Popko, T., I. Peer, R. Shamir, and D. Graur. 2000. A fast algorithm for joint maximum likelihood reconstruction of ancestral amino acid sequences. Molecular Biology and Evolution 17:890–896.

Posada, D. and K. A. Crandall. 1998. Modeltest: testing the model of dna substitution. Bioinformatics 14:817–818.

Powell, M. J. D. 1964. An efficient method of finding the minimum of a function of several variables without calculating derivatives. Computer Journal 7:155–162.

Reeves, J. H. 1992. Heterogeneity in the substaitution process of amino acid sites of protein coded for by mitochondrial. Journal of Molecular Evolution 35:17–31.

Robinson, D. F. and L. R. Foulds. 1981. Comparison of phylogentic trees. Mathmatical Biosciences 53:131–147.

Rodriguez, R., J. L. Oliver, A. Marin, and J. R. Medina. 1990. The general stochastic model of nucleotide substitution. Journal of Theoritical Biology 142:485–501.

Rogers, J. S. and D. L. Swofford. 1998. A fast method for approximating maximum likelihoods of phylogenetic trees from nucleotide sequences. Systematic Biology 47:77–89.

Rohlf, F. J. 1982. Consensus indecies for comparing classifications. Mathmatical Biosciences 59:131–144.

Saitou, N. and M. Nei. 1987. The neighbor-joining method: a new method for reconstructing phylogenetic trees. Molecular Biology and Evolution 4:406–425.

Sanderson, M. J. 1989. Confidence limits on phylogeneties: the bootstap revisited. Cladistics 5:113–129.

Sankoff, D. and R. J. Cedergren. 1983. Simultaneous comparison of three or more sequences related by a tree. Pages 253–263 *in* Time Warps, String Edits, and Macromoleculaes: the Theory and Practice of Sequence Comparison 1st ed. Addison-Wesley: Reading, Mass.

Sankoff, D., R. J. Cedergren, and W. McKay. 1982. A stategy for sequence phylogeny research. Nucleis Acids Research 10:421–431.

Sankoff, D. and P. Rousseau. 1975. Locating the vertices of a steiner tree in an arbitrary space. Mathmatical Programming 9:240–246.

Sattath, S. and A. Tversky. 1977. Additive similarity trees. Psychometrika 42:319–345.

Sidow, A., T. Nguyen, and T. P. Speed. 1992. Estimating the fraction of invariable codons with a capture recapture method. Journal of Molecular Evolution 35:253–260.

Smith, S. A., J. M. Beaulieu, and M. J. Donoghue. 2009. Mega-phylogeny approach for comparative biology: an alternative to supertree and supermatrix approaches. BMC Evolutionary Biology 9:37.

Sokal, R. R. and C. D. Michener. 1958. A statistical method for evaluating systematic relationships. University of Kansas Science Bulletin 38:1409–1438.

Sokal, R. R. and F. J. Rohlf. 1981. Taxonomic congruence in the leptopodomorpha re-examined. Systematic Zoology 30:309–325.

Strimmer, K. and A. von Haeseler. 1996. Quartet puzzling: a quartet maximum-likelihood method for reconstructing tree topologies. Molecular Biology and Evolution 13:964–969.

Sullivan, J. and D. L. Swofford. 2001. Should we use model-based methods for phylogenetic inference when we know assumptions about among-site rate variation and nucleotide substitution pattern are violated? Systematic Biology 50:723–729.

Swofford, D. L. 1991. When are phylogeny estimates from morphology and molecular data incongruent? Pages 295–333 *in* Phylogenetic Analysis of DNA sequences (M. M. Miyamoto and J. Cracraft, eds.) 1st ed. Oxford University Press: New York, N. Y.

Swofford, D. L. and S. H. Berlocher. 1987. Inferring evolutionary trees from gene frequncy data under the principle of maximum parsimony. Systematic Zoology 36:293–325.

Swofford, D. L. and W. P. Maddison. 1987. Reconstructing ancestral character states under wagner parsimony. Mathematical Biosciences 87:199–229.

Swofford, D. L. and W. P. Maddison. 1992. Parsimony, character-state reconstructions, and evolutionary inferences. Pages 186–223 *in* Systematics, historical ecology, and North American freshwater fishes (R. L. Mayden, ed.). Stanford University Press, Stanford.

Swofford, D. L. and G. J. Olsen. 1990. Phylogenetic reconstruction. Pages 411–501 *in* Molecular Systematcs (D. M. Hillis and C. Moritz, eds.) 1st ed. Sinauer Associates, Sunderland, Massachusetts.

Swofford, D. L., G. J. Olsen, P. J. Waddell, and D. M. Hillis. 1996. Phylogenetic inference. Pages 407–514 *in* Molecular Systematics (D. M. Hillis, C. Moritz, and B. K. Mable, eds.) 2nd ed. Sinauer Associates: Sunderland, Massachusetts.

Tajima, F. and M. Nei. 1982. Biases of the estimates of dna divergence obtained by the restriction enzyme technique. Journal of Molecular Evolution 18:115–120.

Tamura, K. and M. Nei. 1993. Estimation of the number of nucleotide substitutions in the control region of mitochondrial dna in humans and chimpanzees. Molecular Biology and Evolution 10:512–526.

Tavare, S. 1986. Some probabilistic and statistical problems on the analysis of dna. Lectures on Mathematics in the Life Sciences 17:57–86.

Templeton, A. R. 1983a. Convergent evolution and non-parametric inferences from restriction fragment dna sequence data. Pages 151–179 *in* Statistical Analysis of DNA Sequence Data (B. Weir, ed.). Marcel Dekker: New York, N. Y.

Templeton, A. R. 1983b. Phylogenetic inference from restriction endonuclease cleavage site maps with particular reference to the evolution of humans and apes. Evolution 37:221–244.

Thompson, E. A. 1975. Human Evolutionary Trees. Cambridge University Press, Cambridge.

Thorne, J. L. and H. Kishino. 1992. Freeing phylogenies from artifacts of alignment. Molecular Biology and Evolution 9:1148–1162.

Thorne, J. L., H. Kishino, and J. Felsenstein. 1991. An evolutionary model for maximum likelihood alignment of dna sequences. Journal of Molecular Evolution 33:114–124.

Thorne, J. L., H. Kishino, and J. Felsenstein. 1992. Inching toward reality - an improved likelihood model of sequence evolution. Journal of Molecular Evolution 34:3–16.

Tillier, E. R. M. and R. A. Collins. 1994. Neighbor-joining and maximum likelihood with rna sequences: addressing the interdependence of sites. Molecular Biology and Evolution 12:7–15.

Waddell, P. J. 1995. Statistical methods of phylogenetic analysis, including Hadamard conjugations, LogDet transforms, and maximum likelihood. Ph.D. thesis Massey University, Palmerston North, New Zealand.

Waddell, P. J., D. Penny, M. D. Hendy, and G. Arnold. 1994. The sampling distribution and covariance matrix of phylogenetic spectra. Molecular Biology and Evolution 11:630–642.

Weir, B. S. 1990. Genetic Data Analysis. Sinauer Associates, Sunderland, MA.

Wiley, E. O. 1981. Phylogenetics: The Theory and Practice of Phylogenetic Systematics. Wiley and Sons: New York, N. Y.

Yang, Z. 1993. Maximum-likelihood estimation of phylogeny from dna sequences when substitution rates differ over sites. Molecular Biology and Evolution 10:1396–1401.

Yang, Z. 1994a. Estimating the pattern of nucleotide substitution. Journal of Molecular Evolution 39:105–111.

Yang, Z. 1994b. Maximum likelihood phylogenetic estimation from dna sequences with variable rates over sites: approximate methods. Journal of Molecular Evolution 39:306–314.

Yang, Z. 1996. Among-site rate variation and its impact on phylogenetic analyses. Trends in Ecology and Evolution 11:367–372).

Yang, Z., N. Goldman, and A. Friday. 1994. Comparison of models for nucleotide substitution used in maximum-likelihood phylogenetic estimation. Molecular Biology and Evolution 11:316–324.

Yang, Z., S. Kumar, and M. Nei. 1995. A new method of inference of ancestral nucleotide and amino acid sequences. Genetics 141:1641–1650.

Zharkikh, A. 1994. Estimation of evolutionary distances between nucleotide sequences. Journal of Molecular Evolution 39:315–329.