8

Phylogeny inference based on parsimony and other methods using PAUP*

THEORY

David L. Swofford and Jack Sullivan

8.1 Introduction

Methods for inferring evolutionary trees can be divided into two broad categories: those that operate on a matrix of discrete characters that assigns one or more attributes or character states to each taxon (i.e. sequence or gene-family member); and those that operate on a matrix of pairwise distances between taxa, with each distance representing an estimate of the amount of divergence between two taxa since they last shared a common ancestor (see Chapter 1). The most commonly employed discrete-character methods used in molecular phylogenetics are ***parsimony*** and ***maximum likelihood*** methods. For molecular data, the character-state matrix is typically an aligned set of DNA or protein sequences, in which the states are the nucleotides A, C, G, and T (i.e. DNA sequences) or symbols representing the 20 common amino acids (i.e. protein sequences); however, other forms of discrete data such as restriction-site presence/absence and gene-order information also may be used.

Parsimony, maximum likelihood, and some distance methods are examples of a broader class of phylogenetic methods that rely on the use of *optimality criteria*. Methods in this class all operate by explicitly defining an *objective function* that returns a score for any input tree topology. This tree score thus allows any two or more trees to be ranked according to the chosen optimality criterion. Ordinarily, phylogenetic inference under **criterion-based methods** couples the selection of

The Phylogenetic Handbook: a Practical Approach to Phylogenetic Analysis and Hypothesis Testing, Philippe Lemey, Marco Salemi, and Anne-Mieke Vandamme (eds.). Published by Cambridge University Press. © Cambridge University Press 2009.

a suitable optimality criterion with a search for an optimal tree topology under that criterion. Because the number of tree topologies grows exponentially with the number of taxa (see Table 1.4), criterion-based methods are necessarily slower than algorithmic approaches, such as ***UPGMA*** or ***neighbor-joining*** (***NJ***), which simply cluster taxa according to a prescribed set of rules and operations (see Chapter 5). However, we believe that criterion-based methods have a strong advantage in that the basis for preferring one tree over another is made mathematically precise, unlike the case for algorithmic methods. For example, NJ was originally described (Saitou & Nei, 1987) as a method for approximating a tree that minimizes the sum of least-squares branch lengths – the ***minimum-evolution*** criterion (see Chapter 5). However, it rarely achieves this goal for data sets of non-trivial size, and rearrangements of the NJ tree that yield a lower minimum-evolution score can usually be found. This result makes it difficult to defend the presentation of an NJ tree as the most reasonable estimate of a phylogeny. With criterion-based methods, it can at least be said that a given tree topology was the best that could be found according to the criterion. If others want to dispute that tree, they are free to criticize the criterion or search for better trees according to the criterion, but it is clear why the tree was chosen. It is true that ***bootstrapping*** or ***jackknifing methods*** (see Chapter 5) can be used to quantify uncertainty regarding the groupings implied by an NJ tree, but fast approximations are also available for criterion-based methods.

Although it contains additional capabilities, the PAUP* program (Swofford, 2002) is primarily a program for estimating phylogenies using criterion-based methods. It includes support for parsimony, maximum likelihood (nucleotide data), and distance methods. This chapter discusses the use of PAUP* as a tool for phylogenetic inference. First, some theoretical background is provided for parsimony analysis, which – unlike distance and maximum likelihood methods (see Chapters 5 and 6) – is not treated elsewhere in this book. Next, strategies are described for searching for optimal trees that are appropriate for any of the existing optimality criteria. Finally, in the Practice section, many of the capabilities of PAUP* are illustrated using the real-data examples common to other chapters in this book.

8.2 Parsimony analysis – background

Although the first widely used methods for inferring phylogenies were pairwise distance methods, parsimony analysis has been the predominant approach used to construct phylogenetic trees from the early 1970s until relatively recently; despite some limitations, it remains an important and useful technique. The basic idea underlying parsimony analysis is simple: one seeks the tree, or collection of trees, that minimizes the amount of evolutionary change (i.e. transformations of one character state into another) required to explain the data (Kluge & Farris, 1969; Farris, 1970; Fitch, 1971).

The goal of minimizing evolutionary change is often defended on philosophical grounds. One line of argument is the notion that, when two hypotheses provide equally valid explanations for a phenomenon, the simpler one should always be preferred. This position is often referred to as "Ockham's Razor": shave away all that is unnecessary. To use simplicity as a justification for parsimony methods in phylogenetics, one must demonstrate a direct relationship between the number of character-state changes required by a tree topology and the complexity of the corresponding hypotheses. The connection is usually made by asserting that each instance of ***homoplasy*** (i.e. sharing of identical character states that cannot be explained by inheritance from the common ancestor of a group of taxa) constitutes an *ad hoc* hypothesis, and that the number of such ad hoc hypotheses should be minimized. Related arguments have focused on the concepts of falsifiability and corroboration most strongly associated with the writings of Karl Popper, suggesting that parsimony is the only method consistent with a hypothetico-deductive framework for hypothesis testing. However, as argued recently by de Queiroz and Poe (2001), careful interpretation of Popper's work does not lead unambiguously to parsimony as the method of choice. Furthermore, the linkage between parsimony and simplicity is tenuous, as highlighted by the recent work of Tuffley and Steel (1997). It demonstrates that parsimony and ***likelihood*** become equivalent under an extremely parameter-rich likelihood model that assigns a separate parameter for each character (site) on every branch of the tree, which is hardly a "simple" model. So, despite more than 20 years of ongoing debate between those who advocate the exclusive use of parsimony methods and those who favor maximum likelihood and related model-based statistical approaches, the issue remains unsettled and the camps highly polarized. Although we place ourselves firmly on the statistical side, we believe that parsimony methods will remain part of a complete phylogenetic-analysis toolkit for some time because they are fast and have been demonstrated to be quite effective in many situations (e.g. Hillis, 1996). In this sense, parsimony represents a useful "fallback" method when model-based methods cannot be used due to computational limitations.

Although parsimony methods are most effective when rates of evolution are slow (i.e. the expected amount of change is low), it is often asserted incorrectly that this is an "assumption" of parsimony methods. In fact, parsimony can perform extremely well under high rates of change as long as the lengths of the branches on the true underlying tree do not exhibit certain kinds of pathological inequalities (Hillis *et al.*, 1994) (see also ***long-branch attraction***, Section 5.3). Nonetheless, it is difficult to state what the assumptions of parsimony analysis actually are. Conditions can be specified in which parsimony does very well in recovering the true evolutionary tree; however, alternative conditions can be found where it fails horribly. For example, in the so-called *Felsenstein zone*, standard parsimony methods converge to the wrong tree with increasing certainty as more data are accumulated. This is because a greater proportion of identical character states will be shared by chance between unrelated taxa than are shared by related taxa due to common ancestry (Felsenstein, 1978; Swofford *et al.*, 1996). Because parsimony requires no explicit assumptions other than the standard one of independence among characters (and some might even argue with that), all one can say is that it assumes that the conditions that would cause it to fail do not apply to the current analysis. Fortunately, these conditions are now relatively well understood. The combined use of model-based methods – less susceptible to long-branch attraction (see Section 5.3) and related artifacts but limited by computational burden – with faster parsimony methods that permit greater exploration of alternative tree topologies provides a mechanism for making inferences maintaining some degree of protection against the circumstances that could cause parsimony estimates alone to be misleading.

8.3 Parsimony analysis – methodology

The problem of finding optimal trees under the parsimony criterion can be separated into two subproblems: (1) determining the amount of character change, or tree length, required by any given tree; and (2) searching over all possible tree topologies for the trees that minimize this length. The first problem is easy and fast, whereas the second is slow due to the extremely large number of possible tree topologies for anything more than a small number of taxa.

8.3.1 Calculating the length of a given tree under the parsimony criterion

For n taxa, an unrooted binary tree (i.e. a fully bifurcating tree) contains n terminal nodes representing those sequences, $n - 2$ internal nodes, and $2n - 3$ branches (edges) that join pairs of nodes. Let τ represent some particular tree topology, which could be, for example, an arbitrarily chosen tree from the space of all possible trees. The length of this tree is given by

$$L(\tau) = \sum_{j=1}^{N} l_j \tag{8.1}$$

where N is the number of sites (characters) in the alignment and l_j is the length for a single site j. This length l_j is the amount of character change implied by a most parsimonious reconstruction that assigns a character state x_{ij} to each node i for each site j. For terminal nodes, the character state assignment is fixed by the input data. Thus, for binary trees:

$$l_j = \sum_{k=1}^{2N-3} c_{a(k),b(k)} \tag{8.2}$$

where $a(k)$ and $b(k)$ are the states assigned to the nodes at either end of branch k, and c_{xy} is the cost associated with the change from state x to state y. In the simplest form of parsimony (Fitch, 1971), this cost is simply 1 if x and y are different or 0 if they are identical. However, other cost schemes may be chosen. For example, one common scheme for nucleotide data is to assign a greater cost to ***transversions*** than to ***transitions*** (see Chapter 1), reflecting the observation that the latter occur more frequently in many genes and, therefore, are accorded less weight. The cost scheme can be represented as a *cost matrix*, or *step matrix*, that assigns a cost for the change between each pair of character states. In general, the cost matrix is symmetric (e.g. $c_{\text{AG}} = c_{\text{GA}}$), with the consequence that the length of the tree is the same regardless of the position of the root. If the cost matrix contains one or more elements for which $c_{xy} \neq c_{yx}$, then different rootings of the tree may imply different lengths, and the search among trees must be done over rooted trees rather than unrooted trees.

Direct algorithms for the determination of the l_j are available and are described briefly in this section. First, however, it will be instructive to examine the calculation of tree length using the brute-force approach of evaluating all possible character-state reconstructions. In general, with an alphabet size of r (e.g. $r = 4$ states for nucleotides, $r = 20$ states for amino acids) and T taxa, the number of these reconstructions for each site is equal to r^{T-2}. Consider the following example:

```
          j
W ... ACAGGAT ...
X ... ACACGCT ...
Y ... GTAAGGT ...
Z ... GCACGAC ...
```

Suppose that the tree ((W, Y),(X, Z)) is being evaluated (see Fig. 5.5 about the NEWICK representation of phylogenetic trees), that the lengths for the first $j - 1$ sites have been calculated, and that the length of site j is to be determined next. Because there are four sequences, the number of reconstructions to be evaluated is $4^{(4-2)} = 16$. The lengths implied by each of these reconstructions under two different cost schemes are shown in Fig. 8.1. With equal costs, the minimum length is two steps, and this length is achievable in three ways (i.e. internal nodes assignment "A–C," "C–C," and "G–C"). If a similar analysis for the other two trees is conducted, both of the trees ((W, X),(Y, Z)) and ((W, Z),(Y, X)) are also found to have lengths of two steps. Thus, this character does not discriminate among

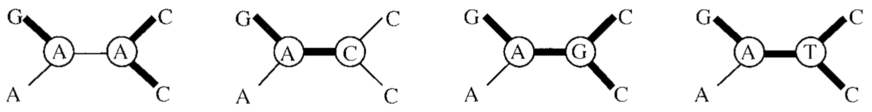
(a)

$$\text{equal} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

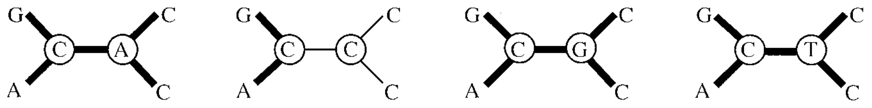
(b)

$$\text{tv4} = \begin{bmatrix} 0 & 4 & 1 & 4 \\ 4 & 0 & 4 & 1 \\ 1 & 4 & 0 & 4 \\ 4 & 1 & 4 & 0 \end{bmatrix}$$

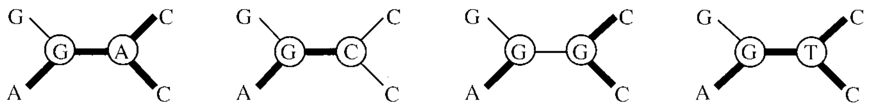
(c)



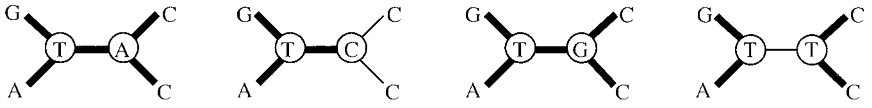
equal: 1+0+0+1+1=3	equal: 1+0+1+0+0=2	equal: 1+0+1+1+1=4	equal: 1+0+1+1+1=3
tv4: 1+0+0+4+4=9	tv4: 1+0+4+0+0=5	tv4: 1+0+1+4+4=10	tv4: 1+0+4+1+1=7



equal: 1+1+1+1+1=5	equal: 1+1+0+0+0=2	equal: 1+1+1+1+1=5	equal: 1+1+1+1+1=5
tv4: 4+4+4+4+4=20	tv4: 4+4+0+0+0=8	tv4: 4+4+4+4+4=20	tv4: 4+4+1+1+1=11



equal: 0+1+1+1+1=4	equal: 0+1+1+0+0=2	equal: 0+1+0+1+1=3	equal: 0+1+1+1+1=4
tv4: 0+1+1+4+4=10	tv4: 0+1+4+0+0=5	tv4: 0+1+0+4+4=9	tv4: 0+1+4+1+1=7



equal: 1+1+1+1+1=5	equal: 1+1+1+0+0=3	equal: 1+1+1+1+1=5	equal: 1+1+0+1+1=4
tv4: 4+4+4+4+4=20	tv4: 4+4+1+0+0=9	tv4: 4+4+4+4+4=20	tv4: 4+4+0+1+1=10

Fig. 8.1 Determination of the length of a tree by brute-force consideration of all possible state assignments to the internal nodes. Calculations are for one site of one of the possible trees for the four taxa W, X, Y, and Z (character states: A, C, G, T). (a) Cost matrix that assigns equal cost to all changes from one nucleotide to another. (b) Cost matrix that assigns four times as much weight to transversions as to transitions (rows and columns are ordered A, C, G, T). (c) All 16 possible combinations of state assignments to the two internal nodes and the lengths under each cost scheme. Minimum lengths are two steps for equal costs and five steps for 4 : 1 tv : ti weighting.

the three tree topologies and is said to be *parsimony-uninformative* under this cost scheme. With 4 : 1 transversion : transition weighting, the minimum length is five steps, achieved by two reconstructions (i.e. internal node assignments "A–C" and "G–C"). However, similar evaluation of the other two trees (not shown) finds a minimum of eight steps on both trees (i.e. two transversions are required rather than one transition plus one transversion). Under these unequal costs, the character becomes informative in the sense that some trees have lower lengths than others, which demonstrates that the use of unequal costs may provide more information for phylogeny reconstruction than equal-cost schemes.

The method used in this example, in principle, could be applied to every site in the data set, summing these lengths to obtain the total tree length for each possible tree, and then choosing the tree that minimizes the total length. Obviously, for real applications, a better way is needed for determining the minimum lengths that does not require evaluation of all r^{n-2} reconstructions. A straightforward ***dynamic programming*** algorithm (Sankoff & Rousseau, 1975) provides such a method for general cost schemes; the methods of Farris (1970), Fitch (1971), and others handle special cases of this general system with simpler calculations. Sankoff's algorithm is illustrated using the example in Box 8.1; the original papers may be consulted for a full description of the algorithm. Dynamic programming operates by solving a set of subproblems and by assembling those solutions in a way that guarantees optimality for the full problem. In this instance, the best length that can be achieved for each subtree – given each of the possible state assignments to each node – is determined, moving from the tips toward the root of the tree. Upon arriving at the root, an optimal solution for the full tree is guaranteed. A nice feature of this algorithm is that, at each stage, only one pair of vectors of conditional subtree lengths above each node needs to be referenced; knowing how those subtree lengths were obtained is unnecessary.

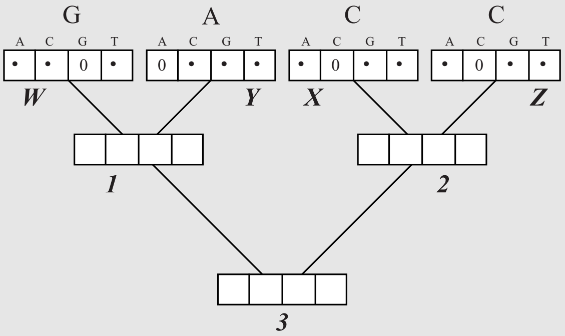
For simple cost schemes, the full dynamic programming algorithm described in Box 8.1 is not necessary. Farris (1970) and Fitch (1971) described algorithms for characters with ordered and unordered states, respectively, which can be proven to yield optimal solutions (Hartigan, 1973; Swofford & Maddison, 1987); these two algorithms are equivalent for binary (i.e. two-state) characters. Fitch's algorithm, illustrated in Box 8.2, is relevant for sequence data when the cost of a change from any state to any other state is assumed to be equal.

8.4 Searching for optimal trees

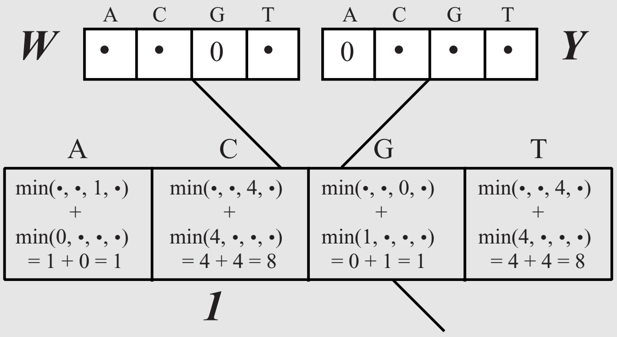
Having specified a means for calculating the score of a tree under our chosen criterion, the more difficult task of searching for an optimal tree can be confronted. The methods described in the following sections can be used for parsimony,

Box 8.1 Calculation of the minimum tree length under general cost schemes using Sankoff's algorithm

For symmetric cost matrixes, we can root an unrooted tree arbitrarily to determine the minimum tree length. Then, for each node *i* (labeled in boldface italics), we compute a conditional-length vector $\mathbf{S}_{ij}$ containing the minimum possible length above *i*, given each of the possible state assignments to this node for character *j* (for simplicity, we drop the second subscript because only one character is considered in turn). Thus, s_{ik} is the minimum possible length of the subtree descending from node *i* if it is assigned state *k*. For the tip sequences, this length is initialized to 0 for the state(s) actually observed in the data or to infinity otherwise. The algorithm proceeds by working from the tips toward the root, filling in the vector at each node based on the values assigned to the node's children (i.e. immediate descendants).



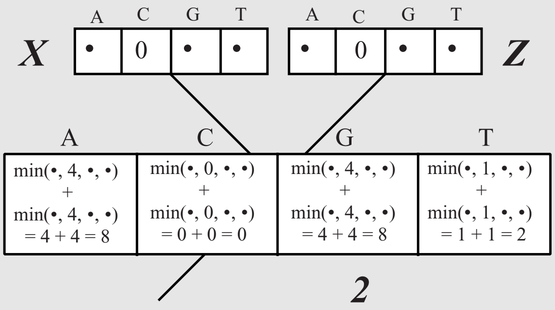
We now visit node 1. For each element *k* of this vector, we consider the costs associated with each of the four possible assignments to each of the child nodes W and Y, and the cost needed to reach these states from state *k*, which is obtained from the cost matrix (**C** in this example, we use the cost matrix from Fig. 8.1B, which represents a 4 : 1 tv : ti weighting). This calculation is trivial for nodes ancestral to two terminal nodes because only one state needs to be considered for each child. Thus, if we assign state A to node 1, the minimum length of the subtree above node 1 given this assignment is the cost of a change from A to G in the left branch, plus the cost of a (non-) change from A to A in the right branch: $s_{1\text{A}} = c_{\text{AG}} + c_{\text{AA}} = 1 + 0 = 1$. Similarly, $s_{1\text{C}}$ is the sum of c_{CG} (left branch) and c_{CA} (right branch), or 8. Continuing in this manner, we obtain



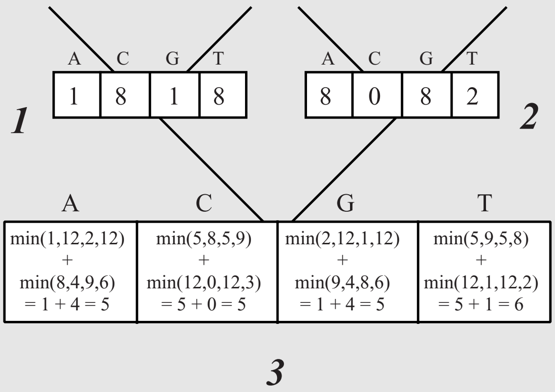
W
Y
A C G T
• • 0 •
0 • • •
A
C
G
T
min(•, •, 1, •) + min(0, •, •, •) = 1 + 0 = 1
min(•, •, 4, •) + min(4, •, •, •) = 4 + 4 = 8
min(•, •, 0, •) + min(1, •, •, •) = 0 + 1 = 1
min(•, •, 4, •) + min(4, •, •, •) = 4 + 4 = 8
1

for the subtree of node 1.

The calculation for node 2 proceeds analogously:


A C G T A C G T
X
• 0 • •
• 0 • •
Z
A C G T
min(•, 4, •, •) + min(•, 4, •, •) = 4 + 4 = 8
min(•, 0, •, •) + min(•, 0, •, •) = 0 + 0 = 0
min(•, 4, •, •) + min(•, 4, •, •) = 4 + 4 = 8
min(•, 1, •, •) + min(•, 1, •, •) = 1 + 1 = 2
2

The calculation for the root node (node 3) is somewhat more complicated: for each state *k* at this node, we must explicitly consider each of the four state assignments to each of the child nodes 1 and 2. For example, when calculating the length conditional on the assignment of state A to node 3, for the left branch we consider in turn all four of the assignments to node 1. If node 1 is assigned state A as well, the length would be the sum of 1 (for the length above node 1) plus 0 (for the non-change from state A to state A). If we instead choose state C for node 1, the length contributed by the left branch would be 8 (for the length above node 1) plus 4 (for the change from A to C). The same procedure is used to determine the conditional lengths for the right branch. By summing these two values for each state *k*, we obtain the entire conditional-length vector for node 3:


A C G T A C G T
1
1 8 1 8
8 0 8 2
2
A C G T
min(1,12,2,12) + min(8,4,9,6) = 1 + 4 = 5
min(5,8,5,9) + min(12,0,12,3) = 5 + 0 = 5
min(2,12,1,12) + min(9,4,8,6) = 1 + 4 = 5
min(5,9,5,8) + min(12,1,12,2) = 5 + 1 = 6
3

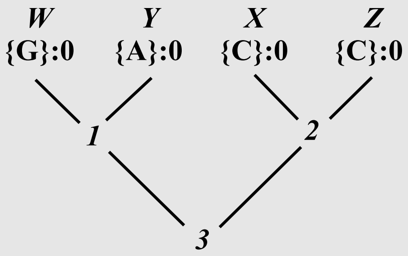
Since we are now at the root of the tree, the conditional-length vector $\mathbf{s}_3$ provides the minimum possible lengths for the full tree given each of the four possible state assignments to the root, and the minimum of these values is the tree length we seek. Observe that this length, 5, is the same value we obtained using the brute-force enumeration shown in Fig. 8.1. As an exercise, the reader may wish to verify that other rootings yield the same length.

This algorithm provides a means of calculating the length required by any character on any tree under any cost scheme. We can obtain the length of a given tree by repeating the procedure outlined here for each character and summing over characters. In principle, we can then find the most parsimonious tree by generating and evaluating all possible trees, although this exhaustive-search strategy would only be feasible for a relatively small number of sequences (i.e. 11 in the current version of PAUP*).

Box 8.2 Calculation of the minimum tree length using Fitch's algorithm for equal costs

As for the general case, we can root an unrooted tree arbitrarily to determine the minimum tree length. We will assign a **state set** X_i to each node i; this set represents the set of states that can be assigned to each node so that the minimum possible length of the subtree above that node can be achieved. Also, for each node, we will store an accumulated length s_i, which represents the minimum possible (unconditional) length in the subtree descending from node i.

The state sets for the terminal nodes are initialized to the states observed in the data, and the accumulated lengths are initialized to zero. For the example of Fig. 8.1, this yields:

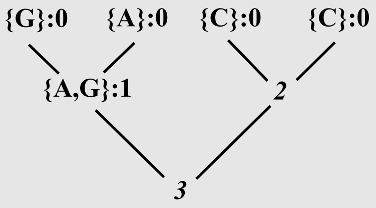


For binary trees, the state-set calculation for each internal node i follows two simple rules, based on the state sets and accumulated lengths of the two descendant child nodes, denoted $L(i)$ and $R(i)$:

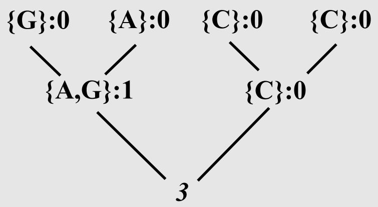
(1) Form the intersection of the two child state sets: $X_{L(i)} \cap X_{R(i)}$. If this intersection is non-empty, let X_i equal this intersection. Set the accumulated length for this node to be the sum of the accumulated lengths for the two child nodes: $s_i = s_{L(i)} + s_{R(i)}$.

(2) If the intersection was empty (i.e. $X_{L(i)}$ and $X_{R(i)}$ are disjoint), let X_i equal the union of the two child state sets: $X_{L(i)} \cup X_{R(i)}$. Set the accumulated length for this node to be the sum of the accumulated lengths for the two child nodes *plus one*: $s_i = s_{L(i)} + s_{R(i)} + 1$.

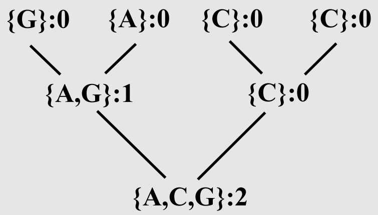
Proceeding to node 1 in our example, we find that the intersection of {G} and {A} is the empty set (ϕ); therefore, by Rule 2, we let $X_1 = \{A\} \cup G = \{A, G\}$ and $s_1 = 0 + 0 + 1 = 1$:



For node 2, the intersection of {C} and {C} is (obviously) {C}; therefore, by Rule 1, we obtain:



Finally, for node 3, the intersection of {A, G} and {C} = ϕ, so the state set at the root is equal to the union of these sets, and the corresponding accumulated length equals $1 + 0 + 1 = 2$:



Thus, the length required by the tree is equal to two steps, as we obtained previously using the brute-force approach. As an exercise, the reader may wish to verify that other rootings of the tree yield the same length.

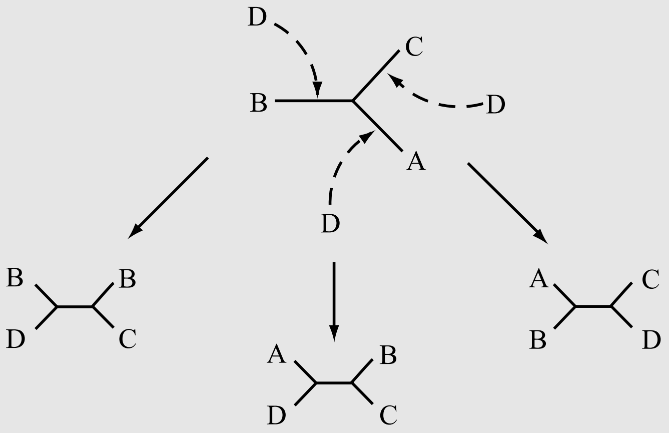
least-squares distance criteria, and maximum likelihood. Regrettably, this search is complicated by the huge number of possible trees for anything more than a small number of taxa.

8.4.1 Exact methods

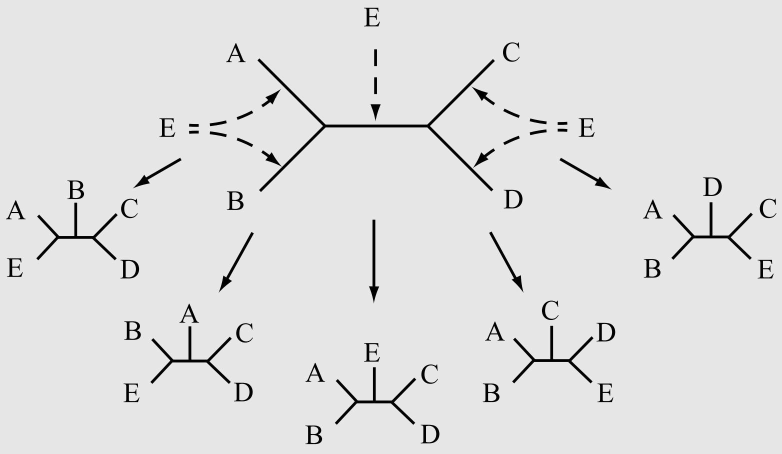
For 11 or fewer taxa, a brute-force **exhaustive search** is feasible; an algorithm is needed that can guarantee generation of all possible trees for evaluation using the previously described methods. The procedure outlined in Box 8.3 is used in PAUP*. This algorithm recursively adds the tth taxon in a stepwise fashion to all possible trees containing the first $t - 1$ taxa until all n taxa have been joined. The algorithm is easily modified for rooted trees by including one additional artificial taxon that locates the root of each tree. In this case, the first three trees generated represent each of the three possible rootings of an unrooted three-taxon tree, and the algorithm proceeds as in the unrooted case. Thus, the number of rooted trees for n taxa is equal to the number of unrooted trees for $n + 1$ taxa.

Box 8.3 Generation of all possible trees

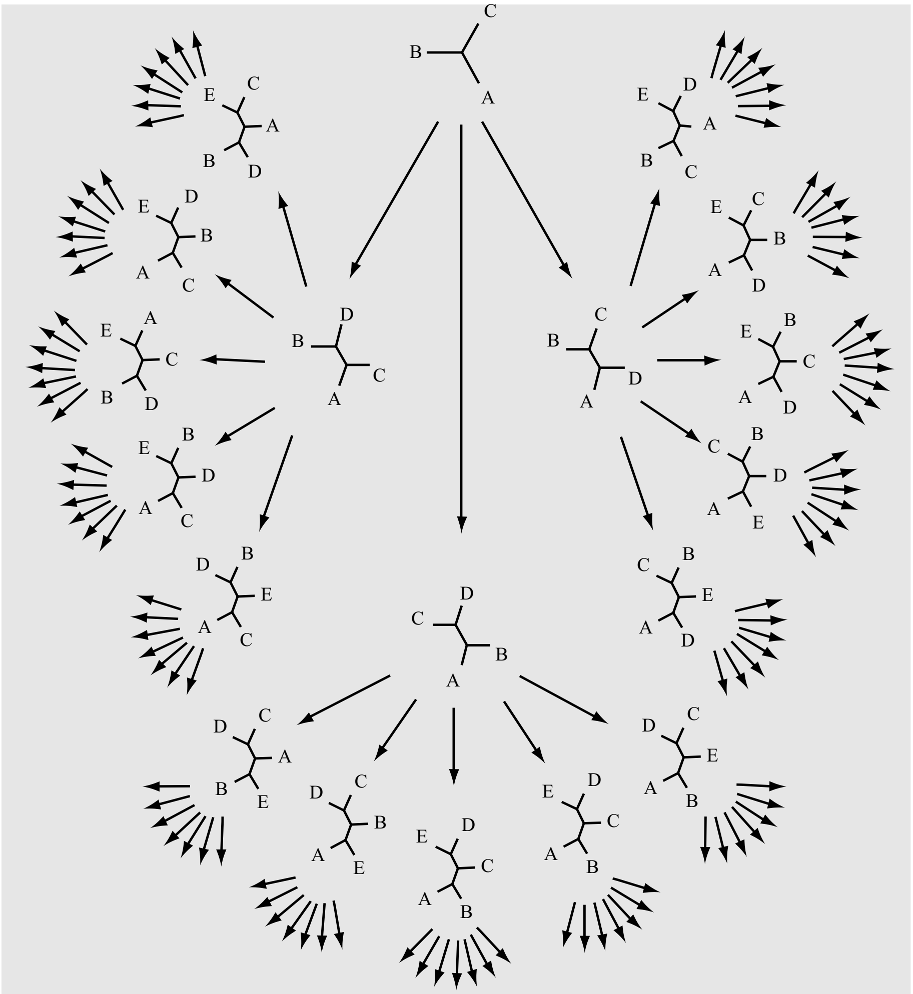
The approach for generation of all possible unrooted trees is straightforward. Suppose that we have a data set containing six sequences (i.e. taxa). We begin with the only tree for the first three taxa in the data set, and connect the fourth taxon to each of the three branches on this tree.



This generates all three of the possible unrooted trees for the first four taxa. Now, we connect the fifth taxon, E, to each branch on each of these three trees. For example, we can join taxon E to the tree on the right above:



By connecting taxon E to the other two trees in a similar manner, we generate all 15 of the possible trees for the first five taxa. Finally, we connect the sixth taxon, F, to all locations on each of these 15 trees (7 branches per tree), yielding a total of 105 trees. Thus, the full-search tree can be represented as follows (only the paths toward the full 6-taxon trees are shown):



The lengths (or likelihoods or distance scores) of each of these 105 trees can now be evaluated, and the set of optimal trees identified.

It is clear from the description of the algorithm for generating all possible trees in Box 8.3 that the number of possible trees grows by a factor that increases by two with each additional taxon, as expressed in the following relationship:

$$B(t) = \prod_{i=3}^{t} (2i - 5) \tag{8.3}$$

where $B(t)$ is the number of unrooted trees for t taxa. For example, the number of unrooted trees for 7 taxa is $1 \times 3 \times 5 \times 7 \times 9 = 945$ and the number of unrooted trees for 20 taxa is over 2×10^{20}. Clearly, the exhaustive-search method can be used for only a relatively small number of taxa. An alternative exact procedure, the ***branch-and-bound method*** (Hendy & Penny, 1982), is useful for data sets containing from 12 to 25 or so taxa, depending on the "messiness" of the data. This method operates by implicitly evaluating all possible trees, but cutting off paths of the search tree when it is determined that they cannot possibly lead to optimal trees. The branch-and-bound method is illustrated for a hypothetical six-taxon data set in Fig. 8.2. We present the example as if parsimony is the optimality criterion, but this choice is not important to the method. The algorithm effectively traces the same route through the search tree as would be used in an exhaustive search (see Box 8.3), but the length of each tree encountered at a node of the search tree is evaluated even if it does not contain the full set of taxa. Throughout this traversal, an upper bound on the length of the optimal tree(s) is maintained; initially, this upper bound can simply be set to infinity. The traversal starts by moving down the left branch of the search tree successively connecting taxa D and E to the initial tree, with lengths of 221 and 234 steps, respectively. Then, connecting taxon F provides the first set of full-tree lengths. After this connection, it is known that a tree of 241 steps exists, although it is not yet known whether this tree is optimal. This number therefore, is taken as a new upper bound on the length of the optimal tree (i.e. the optimal tree length cannot be longer than 241 steps because a tree at this length has already been identified). Now, the algorithm backtracks on the search tree and takes the second path out of the 221-step, 4-taxon tree. The 5-taxon tree containing taxon E obtained by following this path requires 268 steps. Thus, there is no point in evaluating the seven trees produced by connecting taxon F to this tree because they cannot possibly require fewer than 268 steps, and a tree of 241 steps has already been found. By cutting off paths in this way, large portions of the search tree may be avoided and a considerable amount of computation time saved. The algorithm proceeds to traverse the remainder of the search tree, cutting off paths where possible and storing optimal trees when they are found. In this example, a new optimal tree is found at a length of 229 steps, allowing the upper bound on the tree length to be further reduced. Then, when the 233-step tree containing the first five taxa is encountered, the seven trees that would be derived from it can be immediately rejected because they would also require at least 233 steps. The algorithm terminates when the root of the search tree has been visited for the last time, at which time all optimal trees will have been identified.

Several refinements to the branch-and-bound method improve its performance considerably. Briefly, these include (1) using a heuristic method such as ***stepwise addition*** (discussed later in this section) or NJ (see Chapter 5) to find a tree whose length provides a smaller initial upper bound, which allows earlier termination of search paths in the early stages of the algorithm; (2) ordering the sequential addition of taxa in a way that promotes earlier cutoff of paths (rather than just adding them

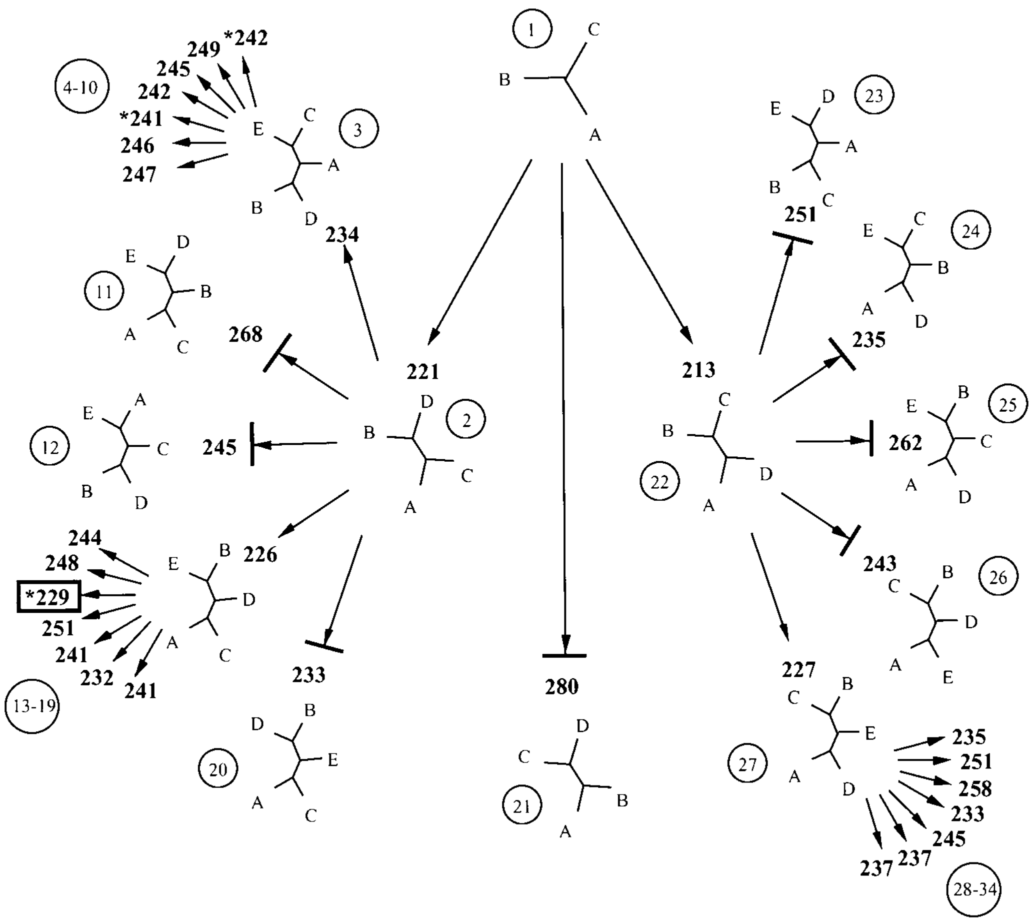


Fig. 8.2 The branch-and-bound algorithm for the exact solution of the problem of finding an optimal parsimony tree. The search tree is the same as shown in Box 8.3, with tree lengths for a hypothetical data set shown in boldface type. If a tree lying at a node of this search tree (thus joining fewer taxa) has a length that exceeds the current lower bound on the optimal tree length, this path of the search tree is terminated (indicated by a cross-bar), and the algorithm backtracks and takes the next available path. When a tip of the search tree is reached (i.e. when we arrive at a tree containing the full set of taxa), the tree is either optimal (and therefore retained) or suboptimal (and rejected). When all paths leading from the initial three-taxon tree have been explored, the algorithm terminates, and all most parsimonious trees will have been identified. Asterisks indicate points at which the current lower bound is reduced. See text for additional explanation; circled numbers represent the order in which phylogenetic trees are visited in the search tree.

in order of their appearance in the data matrix); and (3) using techniques such as pairwise character incompatibility to improve the *lower* bound on the minimum length of trees that can be obtained by continuing outward traversal of the search tree (allowing earlier cutoffs). All of these refinements are implemented in PAUP* (see the program documentation for further information).

The branch-and-bound strategy may be used for any optimality criterion whose objective function is guaranteed to be non-decreasing as additional taxa are connected to the tree. Obviously, this is true for parsimony; increasing the variability of the data by adding additional taxa cannot possibly lead to a decrease in tree length. It is also true for maximum likelihood and some distance criteria, including least-squares methods that score trees by minimizing the discrepancy between observed and path-length distances (see Chapter 5). However, it does not work for the minimum-evolution distance criterion. In minimum evolution, one objective function is optimized for the computation of branch lengths (i.e. least-squares fit), but a different one is used to score the trees (i.e. sum of branch lengths). Unfortunately, the use of these two objective functions makes it possible for the minimum-evolution score to decrease when a new taxon is joined to the tree, invalidating the use of the branch-and-bound method in this case.

8.4.2 Approximate methods

When data sets become too large for the exact searching methods described in the previous section to be used, it becomes necessary to resort to the use of heuristics: approximate methods that attempt to find optimal solutions but provide no guarantee of success. In PAUP* and several other programs, a two-phase system is used to conduct approximate searches. In the simplest case, an initial starting tree is generated using a "greedy" algorithm that builds a tree sequentially according to some set of rules. In general, this tree will not be optimal, because decisions are made in the early stages without regard for their long-term implications. After this starting tree is obtained, it is submitted to a round of perturbations in which neighboring trees in the perturbation scheme are evaluated. If some perturbation yields a better tree according to the optimality criterion, it becomes the new "best" tree and it is, in turn, submitted to a new round of perturbations. This process continues until no better tree can be found in a full round of the perturbation process.

One of the earliest and still most widely used greedy algorithms for obtaining a starting tree is *stepwise addition* (e.g. Farris, 1970), which follows the same kind of search tree as the branch-and-bound method described previously. However, unlike the exact exhaustive enumeration of branch-and-bound methods, stepwise addition commits to a path out of each node on the search tree that looks most promising at the moment, which is not necessarily the path leading to a global optimum. In the example in Fig. 8.2, Tree 22 is shorter than Trees 2 or 21; thus, only trees derivable from Tree 22 remain as candidates. Following this path ultimately leads to selection of a tree of 233 steps (Fig. 8.3), which is only a local rather than a global optimum. The path leading to the optimal 229-step tree was rejected because it appeared less promising at the 4-taxon stage. This tendency to become "stuck" in local optima is a common property of greedy heuristics, and they are often called ***local-search methods*** for that reason.

Because stepwise addition rarely identifies a globally optimal tree topology for real data and any non-trivial number of taxa, other methods must be used to improve the solution. One such class of methods involves tree-rearrangement perturbations known as ***branch-swapping***. These methods all involve cutting off one or more pieces of a tree (subtrees) and reassembling them in a way that is locally different from the original tree. Three kinds of branch-swapping moves are used in PAUP*, as well as in other programs. The simplest type of rearrangement is a ***nearest-neighbor interchange (NNI)***, illustrated in Fig. 8.4. For any binary tree containing T terminal taxa, there are $T - 3$ internal branches. Each branch is visited, and the two topologically distinct rearrangements that can be obtained by swapping a subtree connected to one end of the branch with a subtree connected to the other end of the branch are evaluated. This procedure generates a relatively small number of perturbations whose lengths or scores can be compared to the original tree. A more extensive rearrangement scheme is ***subtree pruning and regrafting (SPR)***, illustrated in Fig. 8.5, which involves clipping off all possible subtrees from the main tree and reinserting them at all possible locations, but avoiding pruning and grafting operations that would generate the same tree redundantly. The most extensive rearrangement strategy available in PAUP* is ***tree bisection and reconnection (TBR)***, illustrated in Fig. 8.6. TBR rearrangements involve cutting a tree into two subtrees by cutting one branch, and then reconnecting the two subtrees by creating a new branch that joins a branch on one subtree to a branch on the other. All possible pairs of branches are tried, again avoiding redundancies.

Note that the set of possible NNIs for a tree is a subset of the possible SPR rearrangements, and that the set of possible SPR rearrangements is, in turn, a subset of the possible TBR rearrangements. For TBR rearrangements, a "reconnection distance" can be defined by numbering the branches from zero starting at the cut branch (Fig. 8.6c and 8.6e). The reconnection distance is then equal to the sum of numbers of the two branches that are reconnected. The reconnection distances then have the following properties: (1) NNIs are the subset of TBRs that have a reconnection distance of 1; (2) SPRs are the subset of TBRs so that exactly one of the two reconnected branches is numbered zero; and (3) TBRs that are neither NNIs nor SPRs are those for which both reconnected branches have non-zero numbers.

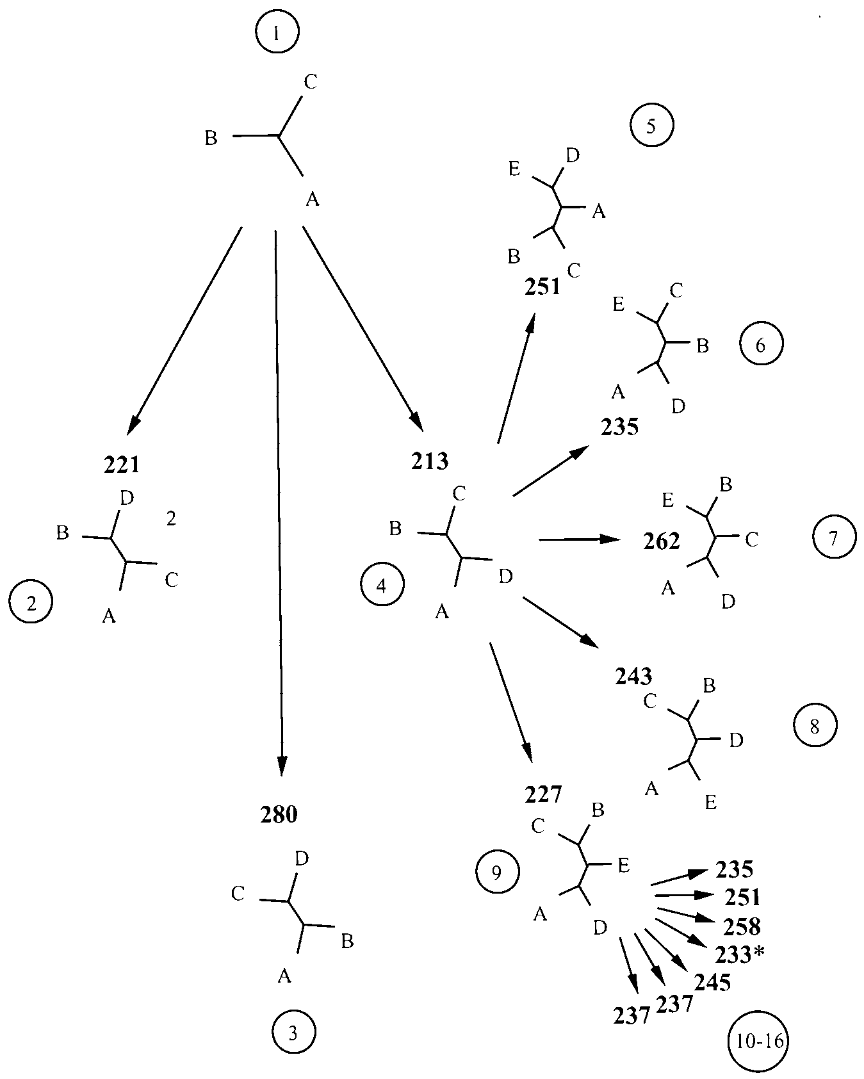


Fig. 8.3 A greedy stepwise-addition search applied to the example in Fig. 8.2. The best four-taxon tree is determined by evaluating the lengths of the three trees obtained by joining Taxon D to Tree 1 containing only the first three taxa. Taxa E and F are then connected to the five and seven possible locations, respectively, on Trees 4 and 9, with only the shortest trees found during each step being used for the next step. In this example, the 233-step tree obtained is not a global optimum (see Fig. 8.2). Circled numbers indicate the order in which phylogenetic trees are evaluated in the stepwise-addition search.

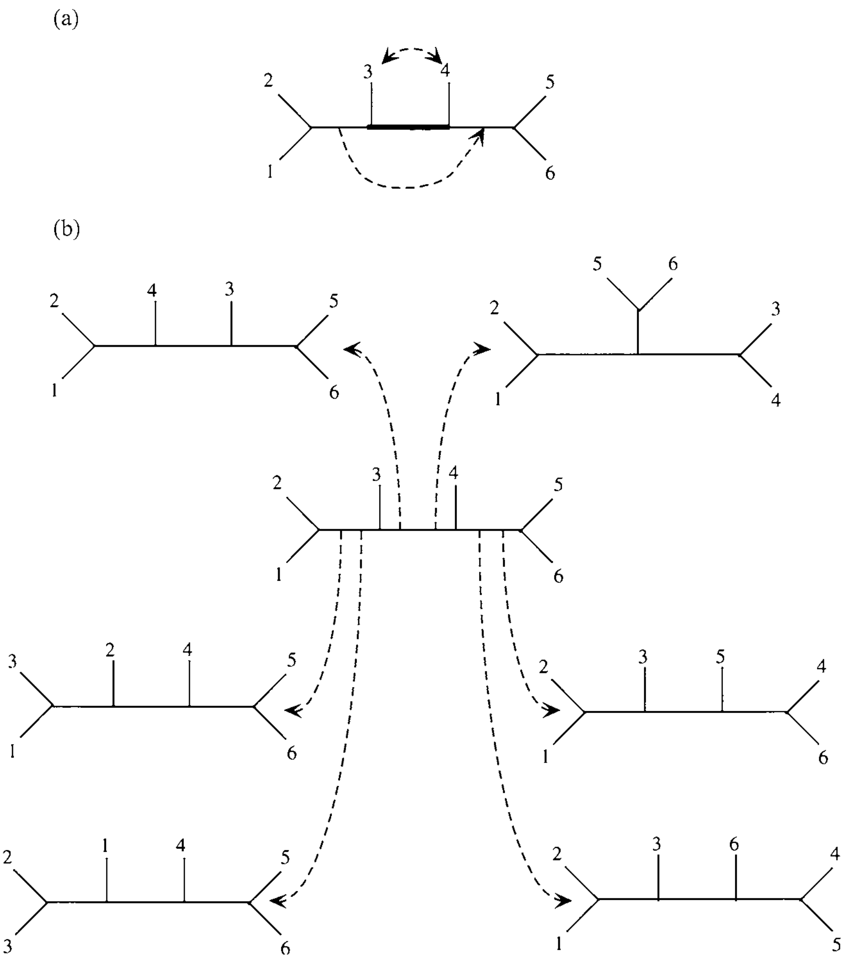


Fig. 8.4 Nearest-neighbor interchange (NNI) rearrangements. (a) An NNI around the central branch. (b) All six of the possible NNIs for the tree in (a).

The reconnection distance can be used to limit the scope of TBR rearrangements tried during the branch-swapping procedure.

The default strategy used for each of these rearrangement methods is to visit branches of the "current" tree in some arbitrary and predefined order. At each branch, all of the non-redundant branch swaps are tried and the score of each resulting tree is obtained (e.g. using the methods described in Box. 8.1 for parsimony). If a rearrangement is successful in finding a shorter tree, the previous tree is discarded and the rearrangement process is restarted on this new tree. If all possible rearrangements have been tried without success in finding a better tree, the swapping process terminates. Optionally, when trees are found that are

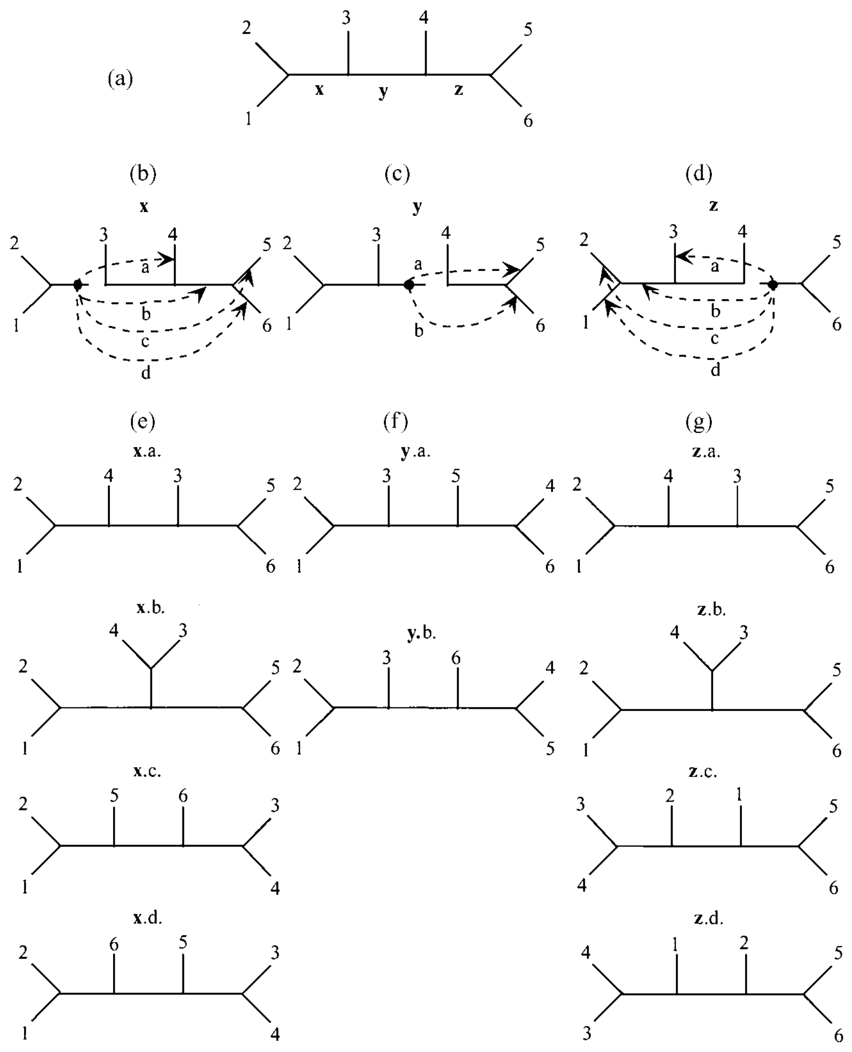


Fig. 8.5 Subtree pruning-regrafting (SPR) rearrangements. (a) A tree to be rearranged. (b), (c), (d) SPRs resulting from pruning of branches **x**, **y**, and **z**, respectively. In addition to these rearrangements, all terminal taxa (i.e. leaves) would be pruned and reinserted elsewhere on the tree. (e), (f), (g) Trees resulting from regrafting of branches **x**, **y**, and **z**, respectively, to other parts of the tree.

equal in score to the current tree (e.g. equally parsimonious trees or trees that have identical likelihoods within round-off error), they are appended to a list of optimal trees. In this case, when the arrangement of one tree finishes, the next tree in the list is obtained and input to the branch-swapping algorithm. If a rearrangement of this next tree yields a better tree than any found so far, all trees in the current list are discarded and the entire process is restarted using the newly discovered

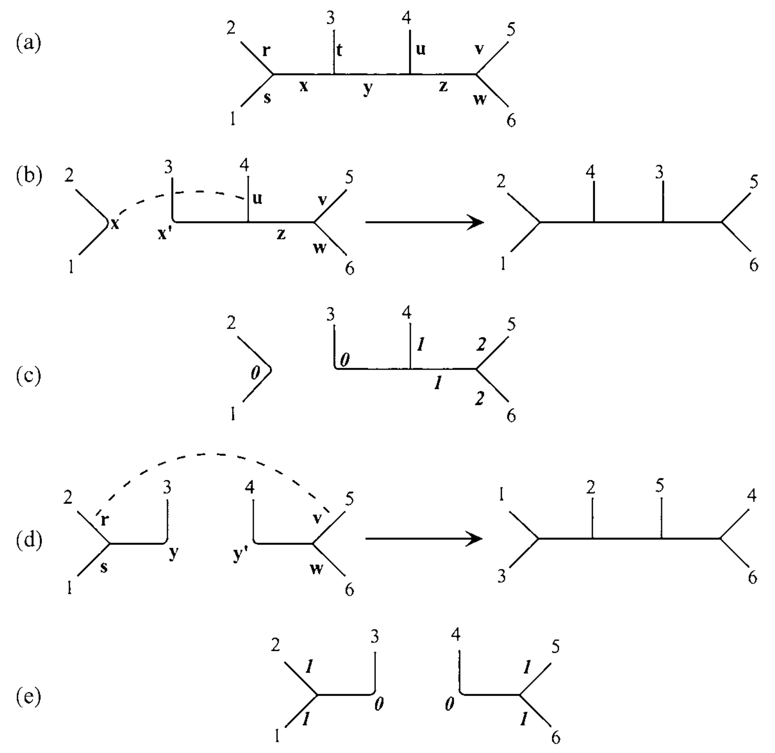


Fig. 8.6 Tree bisection–reconnection (TBR) rearrangements. (a) A tree to be rearranged. (b) Bisection of branch **x** and reconnection to branch **u**; other TBRs would connect **x** to **z**, **v**, and **w**, respectively. (c) Branch-numbering for reconnection distances involving branch **x** (see text). (d) Bisection of branch **y** and reconnection of branch **r** to **v**; other TBRs would connect **r** to **w**, **r** to **y**′, **s** to **v**, **s** to **w**, **s** to **y**′, **y** to **v**, and **y** to **w**, respectively. (e) Branch-numbering for reconnection distances involving branch **y** (see text). All other branches, both internal and external, also would be cut in a full round of TBR swapping.

tree. The algorithm then terminates when every possible rearrangement has been tried on each of the stored trees. In addition to identifying multiple and equally good trees, this strategy often identifies better trees than would be found if only a single tree were stored at any one time. This can happen when all of the trees within one rearrangement of the current tree are no better than the current tree; however, some of the adjacent trees can, in turn, be rearranged to yield trees that are better.

Although they are often quite effective, **hill-climbing algorithms**, such as the branch-swapping methods implemented in PAUP*, are susceptible to the problem of entrapment in local optima. By only accepting proposed rearrangements that are equal to, or better than, the current best tree, these algorithms eventually reach the peak of the slope on which they start; however, the peak may not represent a global optimum. One generally successful method for improving the chances of obtaining a globally optimal solution is to begin the search from a variety of starting points in the hope that at least one of them will climb the right hill. An option available in PAUP* is to start the search from randomly chosen tree topologies. However, in practice, these randomly chosen trees usually fit the data so poorly that they end up merely climbing a foothill on a rugged landscape, and usually fail to find trees that are as good as those resulting from using a starting tree obtained by stepwise addition. An alternative method takes advantage of the fact that, for data sets of non-trivial size and complexity, varying the sequence in which taxa are added during stepwise addition may produce different tree topologies that each fit the data reasonably well. Starting branch-swapping searches from a variety of random-addition-sequence replicates thereby provides a mechanism for performing multiple searches that each begins at a relatively high point on some hill, increasing the probability that the overall search will find an optimal tree.

Random-addition-sequence searches are also useful in identifying multiple "islands" of trees (Maddison, 1991) that may exist. Each island represents all of the trees that can be obtained by a sequence of rearrangements, starting from any tree in the island, keeping and rearranging all optimal trees that are discovered. If two optimal trees exist so that it is impossible to reach one tree by a sequence of rearrangements starting from the other without passing through trees that are suboptimal, these trees are on different islands. Because trees from different islands tend to be topologically dissimilar, it is important to detect multiple islands when they exist.

The methods described previously are generally effective for data sets containing up to 100 or so taxa. However, for larger data sets, they are not as efficient as some newer methods that use a variety of stochastic-search and related algorithms that are better able to avoid entrapment in local optima (Lewis, 1998; Goloboff, 1999; Nixon, 1999; Moilanen, 2001). Nixon's (1999) "parsimony ratchet" can be implemented in PAUP* using the PAUPRAT program (Sikes & Lewis, 2001). In Section 6.4 of Chapter 6, different heuristic approaches in a maximum likelihood framework are discussed.

PRACTICE

David L. Swofford and Jack Sullivan

The basic capabilities of PAUP* (Swofford, 2002; *http://paup.csit.fsu.edu*) are illustrated by analyzing the three different data sets. Because PAUP* does not currently support model-based analyses of amino-acid sequence data, only parsimony analysis and basic tree searching are described using a glycerol-3-phosphate dehydrogenase data set. Distance methods and a variety of additional capabilities are demonstrated using an amniote mitochondrial DNA data set. Finally, the flexibility of PAUP* for evaluating and comparing maximum likelihood models is described through the use of an HIV data set. These data sets are also used in other chapters in this book and are available at *www.thephylogenetichandbook.org*. By performing the three different analyses, we illustrate the convenience and power of being able to switch easily between different optimality criteria in a single program. Versions of PAUP* are available with a full graphical user interface (Macintosh, system 9 or earlier), a partial graphic user interface (Microsoft Windows), and a command–line-only interface (UNIX/Linux, MacOSX, and Microsoft Windows console). Because the command–line interface is available on all platforms, it is used exclusively for the analyses that follow. For each example, it is assumed that PAUP* has been successfully installed and is invoked by typing paup at the operating-system prompt, or double-clicking the executable application in the PAUP* folder (Windows versions). Instructions on how to purchase and install PAUP* for different platforms can be found at *http://paup.csit. fsu.edu*. At the time of writing, great effort is being invested into a major update of the program.

PAUP* uses the NEXUS format for input data files and command scripts. The details of this format, a command-reference manual, and quick-start tutorial describing data-file formats are available at *http://paup.csit.fsu.edu/downl.html*. Box 8.4 covers these topics and outlines how to convert the example data sets from PHYLIP to NEXUS format.

Box 8.4 The PAUP* program

The PAUP* (Phylogenetic Analysis Using Parsimony* and other methods) program is distributed by Sinauer Associates and purchasing information can be found at *http://paup.csit.fsu.edu*. The PAUP* installer creates a PAUP folder on the local computer containing the executable application plus a number of other files, including sample data files in NEXUS format and an extensive documentation in pdf format. Versions of the program are available for Macintosh, MS Windows, and UNIX/Linux. The Mac version is controlled using either a full graphical interface (MacOS 9 or earlier) or a command–line interface; only the command–line interface is available for the other platforms. At the time of writing, however, graphical interfaces are being developed for all platforms. Because it is intuitively easy to switch to the Mac graphic interface for the user familiar with the PAUP* commands, this book focuses on how to use the program through the command–line interface. What follows is a brief introduction to the NEXUS format and the main PAUP* commands. More details can be found in the quick-start tutorial available at *http://paup.csit.fsu.edu/downl.html* and in the pdf documentation in the PAUP folder.

The NEXUS format

PAUP* reads input files in NEXUS format. A NEXUS file must begin with the string
#NEXUS
on the first line. The actual sequences plus some additional information about the data set are then written in the next lines within a so-called data block. A data block begins with the line
Begin data;
and it must end with the line
End;
Here is an example of a NEXUS file containing four sequences ten nucleotides long:
(`Dimensions ntax = 4 nchar = 10`):

```
#NEXUS
      Begin data;
      Dimensions ntax = 4 nchar = 10;
      Format datatype = nucleotide gap = – missing = ? interleave;
      Matrix
L20571  ATGACAG-AA
AF10138 ATGAAAG-AT
X52154  AT?AAAGTAT
U09127  ATGA??GTAT
;
End;
```

The sequences are DNA with gaps indicated by a – symbol, and missing characters indicated by a ? symbol (`format datatype = nucleotide gap = – missing = ? matchchar = . Interleave`). The aligned sequences are reported on the line after the keyword `Matrix`. Each sequence starts on a new line with its name followed by a few blank spaces and then the nucleotide (or amino-acid) sequence itself. A semicolon (`;`) must be placed in the line after the last sequence (to indicate the end of the `Matrix` block).

Running PAUP* (MS Windows version)

By double-clicking the PAUP executable inside the PAUP folder (the version available at the time of this writing is called win-paup4b10.exe), an Open window appears asking to select a file to be edited or executed. By choosing edit, the file is opened in a text-editor window, so that it is possible, for example, to convert its format to NEXUS by manual editing. If the file is already in NEXUS format, it can be executed by selecting Execute in the Open window. The PAUP* `Display` window appears showing information about the data in the NEXUS file (e.g. number of sequences, number of characters) which are now ready to be analyzed by entering PAUP* commands in the command–line at the bottom of the PAUP* Display window. By clicking on cancel, an empty PAUP* `Display` window appears. To quit the program, just click the `x` in the upper right corner of the PAUP* `Display` window, or choose exit from the File menu.

During a phylogenetic analysis with PAUP*, it is good practice to keep a record of all the different steps, displayed in the program Display window, in a log file. To do that at the beginning of a new analysis, choose `Log output to disk` from the `File` menu. The log file can be viewed later with any text editor, including PAUP* working in edit mode (described previously). The program can convert aligned sequences from PHYLIP and other formats to NEXUS. In the following example, the file `hivALN.phy` (available at *www.thephylogenetichandbook.org*) is converted from PHYLIP to NEXUS format:

(1) Place the file `hivALN.phy` in the PAUP folder.
(2) Run PAUP* by double-clicking the executable icon and click on cancel.
(3) Type in the command–line the following:
(4) `toNexus fromFile = hivALN.phy toFile = hivALN.nex;`
(5) Click `execute` below the command–line or press `Enter.`

A file called `hivALN.nex` will appear in the PAUP folder.

The command `toNexus` converts files to the NEXUS format. The option `fromFile =` indicates the input file with the sequences in format other than NEXUS. `toFile =` is used to name the new NEXUS file being created. Note that PAUP* is case-insensitive, which means that, for example, the commands `toNexus` and `TONEXUS` are exactly the same. In what follows, uppercase and lowercase in commands or options are only used for clarity reasons. The file `gdp.phy`, containing the protein example data set (available at *www.thephylogenetichandbook.org*) in PHYLIP interleaved format (see Box 2.4), is translated into NEXUS with the following command block:

```
toNexus fromFile = gdp.phy toFile = gdp.nex Datatype = protein
Interleaved = yes;
```

Note that it is necessary to specify protein for the option `Datatype` and to set `Interleaved` to `yes` (the starting default option is no).

Commands and options

As discussed in this chapter, PAUP* can perform distance-, parsimony-, and maximum-likelihood-based phylogenetic analyses of molecular sequence data. Such analyses are carried out by entering commands on the command–line. For most commands, one or more options may be supplied. Each option has a default value, but the user can assign an alternative value among those allowed for that particular option by typing `option = value`. Any command and its options with assigned values can be typed in the command–line at the bottom of the PAUP* `Display` window and executed by clicking `execute` below the command–line or pressing `Enter`. A semicolon (;) is used to indicate the end of a particular command. If desired, multiple commands can be entered, each separated from the next by a semicolon, in which case the commands are executed in the order they appear in the command–line. The complete list of available options for a given command can be obtained by entering

`command—name` ?;

in the PAUP* command–line, where *command-name* represents the name of a valid PAUP* command. In this way, it is also possible for the user to check the current value of each option. Try, for example, to type and execute `Set ?` (when only one command is executed, the semicolon can be omitted). The following list will be displayed in the PAUP* window:

```
Usage: Set [options ...];
Available options:
Keyword   —— Option type ————————————— Current default-
Criterion    Parsimony|Likelihood|Distance Parsimony
MaxTrees      <integer-value> 100
```

This list is an abridged version of the possible options and their current values for the `Set` command. `Set` is used to choose whether to perform a parsimony-, maximum-likelihood-, or distance-based analysis by assigning the appropriate value – selected among the `Option` type in the list – to the `Criterion` option. By default, PAUP* performs parsimony-based analysis. By typing `set Criterion=Likelihood or set Criterion=Distance` in the command line, the analysis criterion switches to maximum likelihood or distance, respectively.

`MaxTrees` is another option that can be used in conjunction with the `Set` command. `MaxTrees` controls the maximum number of trees that will be saved by PAUP* when a heuristic tree search with parsimony or maximum likelihood is performed (see Sections 8.4.2 and 8.6). The default value is 100, but the user can enter a different integer number (e.g. 1000) by typing `set MaxTrees = 1000`.

8.5 Analyzing data with PAUP* through the command–line interface

A typical phylogenetic analysis with PAUP* might follow these steps:

(1) Choose a phylogenetic optimality criterion: parsimony (default when the program starts), maximum likelihood (Set criterion=likelihood) or distance `Set criterion=distance`).
(2) Choose the appropriate settings for the selected parsimony, likelihood, or distance analysis with the `PSet`, `LSet`, or `DSet` command, respectively.
(3) Compute the tree topology with the HSearch (all optimality criteria) or NJ (distance) command (see Section 8.7).
(4) Evaluate the reliability of the inferred tree(s) with bootstrapping (`Bootstrap` command). Also, for maximum likelihood, the zero-branch-length test is available (see Section 8.8).
(5) Examine the tree in the PAUP* `Display` window with the `DescribeTrees` command; or save it in a tree file with the `SaveTrees` command by typing `savetrees file=`*`tree − file − name`*).

A note of caution: By default, the `SaveTrees` command saves trees in NEXUS format without including branch lengths. To include branch lengths, type:

```
savetrees file=tree-file-name brlens=yes;
```

Trees in NEXUS format can be opened with TREEVIEW of FIGTREE (see Chapter 5) or reimported in PAUP* (after executing the original sequence data file) with the following command:

```
gettrees file=tree-file-name;
```

Complete details can be found in the pdf manual in the PAUP folder.

8.6 Basic parsimony analysis and tree-searching

Assuming that the file gpd.nex is located in the current directory, start PAUP* and execute the file (execute gpd.nex). The number of taxa in this data set (i.e. 12) is effectively the maximum for which an exhaustive search is feasible. This can be accomplished using the command:

```
alltrees fd=histogram;
```

The fd=histogram option requests output of the frequency distribution of tree lengths in the form of a histogram with the default number (i.e. 20) of class intervals. The resulting output follows:

```
Exhaustive search completed:
  Number of trees evaluated = 654729075
  Score of best tree found = 812
  Score of worst tree found = 1207
  Number of trees retained = 1
  Time used = 00:59:02 (CPU time = 00:57:27.8)

Frequency distribution of tree scores:

             mean = 1108.333294 sd = 48.441980
               g1 = -0.817646   g2 = 0.531509

 812.00000 +-------------------------------------
 831.75000 | (337)
 851.50000 | (4337)
 871.25000 | (26850)
 891.00000 | (96474)
 910.75000 | (305244)
 930.50000 | (761800)
 950.25000 |# (1797170)
 970.00000 |## (3373829)
 989.75000 |#### (6937378)
1009.50000 |######## (12789821)
1029.25000 |############# (21003098)
1049.00000 |################## (32167531)
1068.75000 |###################### (49814274)
1088.50000 |######################## (70216478)
1108.25000 |########################### (89088850)
1128.00000 |############################### (98980066)
1147.75000 |################################### (114383861)
1167.50000 |################################ (103567496)
1187.25000 |############################# (46015542)
1207.00000 |## (3398639)
          +-------------------------------------
```

Thus, it is known with certainty that the shortest tree requires 812 steps and that there is only one tree of this length. Furthermore, although there are many trees slightly longer than the shortest tree, these "good" trees represent only a small fraction of the more than 6.5×10^8 possible trees for 12 taxa. This search requires significant computation time (i.e. approximately 1 hour on a moderately fast Intel Core 2 Duo Powerbook). If the distribution of tree lengths is not of interest, the branch-and-bound algorithm may be used to find an exact solution more quickly by typing:

```
bandb;
```

This search takes less than 1 second and finds the same MP tree. If the data set had been much larger, finding exact solutions using the exhaustive-search or branch-and-bound methods would not have been feasible, in which case the `HSearch` command could be used to perform a heuristic search. For example,

```
hsearch/addseq=random nreps=100;
```

would request a search from 100 starting trees generated by stepwise addition with random-addition sequences, followed by TBR (the default) branch-swapping on each starting tree. In this small example, the majority of the random-addition-sequence replicates finds the best tree obtained using the exact methods, but a small percentage of random-addition-sequence replicates finds a tree that requires one additional evolutionary change. These two trees are different islands.

To examine the trees, the `ShowTrees` or `DescribeTrees` commands are available. `ShowTrees` simply outputs a diagram representing each tree topology, but provides no other information:

```
showtrees all;
```

The best tree is shown in the output as follows:

```
Tree number 1 (rooted using default outgroup)
/---------------------------------------- gpd1yeast
|
|         /------------------------------ gpdadrome
|         |
|         |                    /-------- gpdhuman
|         |              /-----+
+---------+              |     \-------- gpdarabit
|         |       /------+
|         |       |      \-------------- gpdamouse
|         \-------+
|                 \--------------------- gpdacaeel
|
|                              /-------- gpdleish
|                /-------------+
|                |             \-------- gpdtrybb
|                |
|        /-------+             /-------- gpdaecoli
|        |       |      /------+
|        |       |      |      \-------- gpdahaein
\--------+       \------+
         |              \--------------- gpdpseu
         |
         \------------------------------ gpdabacsu
```

Remember that, in general, parsimony analysis does not determine the location of the root. By default, PAUP* assumes that the first taxon in the data matrix is the only ***outgroup*** taxon and roots the tree at the adjacent internal node leading to a basal ***polytomy*** as indicated previously. In this example, the root should probably be located between the cluster of bacterial (gpdaecoli, gpdahaein, gpdabacsu and gpdpseu) and Trypanosomatidae (gpdleish, gpdtrybb) sequences and the cluster of other eukaryote homologues. This can be achieved by specifying the taxa of one cluster as the outgroup, for example:

```
outgroup gpdleish gpdtrybb gpdaecoli gpdahaein gpdabacsu
gpdpseu;
```

or, alternatively, by

```
outgroup 7-12;
```

The tree may now be output according to the new outgroup specification by reissuing the `ShowTrees` command. Because this will not display the ingroup as ***monophyletic***, it is advisable to also specify:

```
Set outroot=mono;
```

Instead of `ShowTrees`, the `DescribeTrees` command can be used to request output of the tree in ***phylogram*** format, where the branches of the tree are drawn proportionally to the number of changes (see Chapter 5) assigned under the parsimony criterion. Other information is also available through additional `DescribeTrees` options. For example, the command

```
describetrees 1/plot=phylogram diagnose;
```

outputs a drawing of the best tree in phylogram format (now reflecting the new rooting), plus a table showing the number of changes required by each character on the first tree, the minimum and maximum possible number of changes on any tree, and several goodness-of-fit measures derived from the following values:

```
Tree number 1 (rooted using user-specified outgroup)

 Tree length = 812
 Consistency index (CI) = 0.8300
 Homoplasy index (HI) = 0.1700
 CI excluding uninformative characters = 0.8219
 HI excluding uninformative characters = 0.1781
 Retention index (RI) = 0.7459
 Rescaled consistency index (RC) = 0.6191

 /---------------------------------- gpd1yeast
 |
 |               /------------ gpdadrome
 22              |
 ||              |                /-- gpdhuman
 ||              |             /-13
 |\-------------16             |  \---- gpdarabit
 |               |     /------14
 |               |     |       \-- gpdamouse
 |               \----15
 |                     \----------- gpdacaeel
 |
 |                  /--------- gpdleish
 |       /---------17
 |       |          \--------- gpdtrybb
 |       |
 |  /---20              /----- gpdaecoli
 |  |    |          /--------18
 |  |    |          |         \------ gpdahaein
 \-21    \----------19
 |                   \---------- gpdpseu
 |
 \------------------ gpdabacsu

 Character diagnostics:
                  Min  Tree   Max                                    G-
 Character Range steps steps steps     CI      RI      RC      HI    fit
 ----------------------------------------------------------------------
 1             2     2     2     6  1.000   1.000   1.000   0.000   1.000
 3             5     5     5     6  1.000   1.000   1.000   0.000   1.000
 .             .     .     .     .   .       .       .       .       .
 .             .     .     .     .   .       .       .       .       .
 .             .     .     .     .   .       .       .       .       .
 233           6     6     6     7  1.000   1.000   1.000   0.000   1.000
 234           1     1     2     2  0.500   0.000   0.000   0.500   0.750
 (24 constant characters not shown)
```

To assess whether this result is strongly supported, bootstrap or jackknife analysis (see Chapter 5) can be performed using the `Bootstrap` or `Jackknife` commands. For example, the command

```
bootstrap nreps=1000;
```

requests a bootstrap analysis with 1000 pseudoreplicate samples using the current optimality criterion and associated settings (`hsearch/addseq=random nreps=100`).

Results of the bootstrap analysis can be summarized using a ***majority-rule consensus tree*** that indicates the frequency in which each bipartition, or division of the taxa into two groups, was present in the trees obtained for each bootstrap replicate (see Box 5.3). Interpretation of *bootstrap values* is complicated (e.g. Hillis & Bull, 1993) (see also Chapter 5), but clearly clades with close to 100% support can be considered very robust. When the bootstrap analysis is completed, the majority-rule consensus with bootstrap values is shown:

```
Bootstrap 50% majority-rule consensus tree

             /-------------------------- gpd1yeast(1)
             |
             |         /---------------- gpdadrome(2)
/------------+         |
|            |         |                    /------ gpdhuman(3)
|            |         |         /----75----+
|            \---100---+         |          \------ gpdarabit(5)
|                      +---100---+
|                      |         \----------------- gpdamouse(4)
|                      |
|                      \--------------------------- gpdacaeel(6)
100
|                                           /------ gpdleish(7)
|                      /---------100--------+
|                      |                    \------ gpdtrybb(8)
|                      |
|                      |                    /------ gpdaecoli(9)
|                      |          /---100---+
\----------------------+          |         \------ gpdahaein(10)
                       +----59----+
                       |          \---------------- gpdpseu(12)
                       |
                       \--------------------------- gpdabacsu(11)
```

In addition to the consensus tree, the output includes a table showing the frequency of all groups that were found at a frequency of 5% or higher, so that support for groups not included in the consensus also can be evaluated as follows:

```
Bipartitions found in one or more trees and frequency of occur-
rence (bootstrap support values):
1 1
123456789012     Freq       %
-----------------------------
........**..  1000.00  100.0%
..***.......  1000.00  100.0%
......******  1000.00  100.0%
......**....   999.67  100.0%
.*****......   997.07   99.7%
..*.*.......   747.89   74.8%
........**.*   589.53   59.0%
.*...*......   498.14   49.8%
......****.*   476.07   47.6%
..****......   407.86   40.8%
........****   313.48   31.3%
........***.   230.47   23.0%
..**........   177.42   17.7%
......*****.   135.95   13.6%
.****.......    91.80    9.2%
..........**    77.38    7.7%
...**.......    74.70    7.5%
......****..    67.33    6.7%
......**...*    59.53    6.0%
```

Note: the format of this output follows the same convention of the output file from the program `Consense.exe` of the PHYLIP package; see Box 5.4). Another measure of nodal support is the ***decay index***, or ***Bremer support*** (Bremer, 1988; Donoghue *et al.*, 1992). If the shortest tree inconsistent with the monophyly of some group is three steps longer than the most parsimonious tree (on which the group is monophyletic), the decay index is equal to 3. These values can be calculated in two ways. For groups that decay quickly, all trees within one step, two steps, and so on, of the shortest tree can be stored and a strict consensus tree calculated for each length. The following commands can be used to find groups that occur in all trees within 1, 2, and 3 steps, respectively, of the shortest tree:

```
hsearch keep=813; contree;
hsearch keep=814; contree;
hsearch keep=815; contree;
```

For example, a group consisting of human (`gpdhuman`) and rabbit (`gpdarabit`) is found in all trees of length 813 and shorter, but this group disappears from the consensus of all trees of length 814 and shorter, indicating that there are some trees at 814 steps that do not contain this 2-taxon clade. The decay index is therefore, 814 – 812 = 2. Evaluating the meaning of decay indexes is probably even more difficult than bootstrap values, but any group for which the decay index is less than 4 should probably be viewed with suspicion (DeBry, 2001).

An alternative method for calculating decay indexes is to use the *converse constraints* feature of PAUP*. In general, a *constraint tree* is a user-defined tree that limits the search space to those trees that are either compatible or incompatible with the constraint tree. For monophyly constraints, a tree *T* is compatible with the constraint tree if it is equal to the constraint tree or if the constraint tree can be obtained by collapsing one or more branches of *T*. For calculating decay indexes, the monophyly of the group of interest is indicated and a search is performed using converse constraints. For the example discussed previously, the constraint tree can be defined using the command

```
constraints humanrabbit=((3,5));
```

The tree resulting from this definition can then be viewed using the `showconstr` command:

```
Constraint-tree "humanrabbit":
/--------------------------------------- gpd1yeast
|
+--------------------------------------- gpdadrome
|
|            /-------------------------- gpdhuman
+------------+
|            \-------------------------- gpdarabit
|
+--------------------------------------- gpdamouse
|
+--------------------------------------- gpdacaeel
|
+--------------------------------------- gpdleish
|
+--------------------------------------- gpdtrybb
|
+--------------------------------------- gpdaecoli
|
+--------------------------------------- gpdahaein
|
+--------------------------------------- gpdabacsu
|
\--------------------------------------- gpdpseu
```

Next, a constrained search is conducted as follows:

```
HSearch enforce constraints=humanrabbit converse addseq=random
nreps=100;
```

It is especially desirable to use multiple random-addition-sequence starting points when searching under converse constraints to improve the chances of identifying the shortest trees that are incompatible with the constraint tree. In this case, five trees of length 814 are found in the constrained search, yielding a decay index of 2, as was determined previously. This second approach is much more useful for determining larger decay values, where saving all trees within *n* steps of the shortest tree becomes more unwieldy as *n* increases. As is often but not always the case, the modest decay index in the example mirrors a modest bootstrap value for the same group.

8.7 Analysis using distance methods

The same search strategies discussed previously may be used for searching trees under distance optimality criteria. Because distance methods were covered extensively in Chapter 5, what follows only illustrates how to perform some commonly used analyses with PAUP*, using the vertebrate mtDNA data set as an example.

The `DSet` command is used to set the type of distance to be computed:

```
dset distance=jc;
```

This will set the distances to Jukes–Cantor distances (see Chapter 4); other distance options available in PAUP* (as well as the current settings) can be viewed by typing `Dset?`. To calculate a standard NJ tree (Saitou & Nei, 1987) according to the current distance transformation, use the command

```
nj;
```

Before proceeding with further analyses, note that the vertebrate mtDNA data set exhibits heterogeneous base frequencies among taxa, as can be seen from the output of the command

```
basefreq;
```

In the presence of such heterogeneity, an appropriate distance transformation is the LogDet distance (see Section 4.13), which can be selected using the command

```
dset distance=logdet;
```

Criterion-based searches are performed using the same commands as were previously described for parsimony analysis. However, because the default optimality criterion is parsimony, the following command must be issued to force use of distance-based optimality criteria.

```
set criterion=distance;
```

By default, distance analyses use an NJ method to obtain a starting tree (`addseq =nj`). This can be overridden to use random addition sequences, as follows:

```
hsearch/start=stepwise addseq=random nreps=100;
```

The resulting tree can then be output as a phylogram:

```
describe/plot=phylogram;
```

```
Minimum evolution score = 2.47415

/-------------------- LngfishAu
|
|                /--------- LngfishSA
|   /-----------18
|   |            \--------- LngfishAf
|   |
|   |            /--------- Frog
|   |            |
|   |            |    /------------ Turtle
\--32            |    |
    |            20   /---------------- Crocodile
    |            |\---19
    |            |    \------------ Bird
    |        /---22
    \---31   |   |/--------------- Sphenodon
         |       21
         |        \----------------- Lizard
         |
         |             /---------- Human
         |             |
         |            /25  /------- Seal
         \----30      | |  |
               |      |\-24  /------- Cow
               |      |   \-23
               |  /--27      \------- Whale
               |  |   |
               |  |   |      /----- Mouse
               |  |   \-----26
               \--29         \------ Rat
                  |
                  |  /------------ Platypus
                  \-28
                     \---------- Opossum
```

This tree places turtles with archosaurs. Constraints may be used to find the best tree consistent with the conventional placement of turtle (i.e. basal reptile):

```
constraints turtlebasal = ((Crocodile, Bird, Sphenodon, Lizard),
Turtle);
```

If you previously enforced converse constraints, this option will persist in the current PAUP* session unless you specify `converse = no`:

```
hsearch enforce converse=no constraints=turtlebasal;
```

The score of the resulting tree is 2.47570, which is barely worse than the best unconstrained tree. The low support for the placement of the turtle can be confirmed using bootstrap analysis:

```
bootstrap nreps=1000 / enforce=no;
```

The bootstrap support results, in fact, in only about 51% (this value can slightly vary because of the stochastic nature of bootstrap resampling).

Minimum evolution (see Chapter 5) is the default method used for distance analyses in PAUP*. This method uses least-squares fit to determine the branch lengths, but then uses the sum of these branch lengths as the objective function for computing the score of each tree. The unweighted least-squares and Fitch–Margoliash (weighted least-squares) variants are also available by replacing *n* with 0 or 2, respectively, in the following command

```
dset objective=ls power=n;
```

prior to beginning a distance-criterion search.

One particularly problematic aspect of distance analysis involves the handling of negative branch lengths, which often occur in unconstrained optimization of the least-squares fit between path-length and observed distances. Because negative branch lengths have no obvious biological meaning, they probably should be avoided. A constrained optimization is available in PAUP* to perform the least-squares optimization under the restriction of branch-length non-negativity, which can be requested as follows:

```
dset negbrlen=prohibit;
```

We believe that this option provides the most appropriate way to deal with the negative-branch length problem, but analyses run significantly more slowly when this option is specified. The default setting in PAUP* is to allow negative branch lengths when performing the least-squares fit calculations, but to set negative values to zero before computing the score of the tree (`negbrlen = setzero`). Thus, if the least-squares fit does not require negative lengths, the tree scores obtained in this way are identical, whereas trees for which the fit is improved by allowing negative branch lengths are still penalized accordingly. Negative branch lengths also may be set to their absolute value (i.e., `negbrlen = setabsval`) (Kidd & Sgaramella-Zonta, 1971) or simply used "as is" (i.e. `negbrlen = allow`) (Rzhetsky & Nei, 1992).

It is also possible to input distances directly into PAUP* using a NEXUS format Distances block. This feature permits mixing the tree-searching capabilities of PAUP* with distance transformations that are not available within the program (e.g. model-based protein distances, but their implementation is in development at the time of writing). Examples for using user-supplied distance matrixes are given in the command-reference document in the PAUP folder.

8.8 Analysis using maximum likelihood methods

Maximum likelihood methods are discussed in Chapter 6, including the ***quartet-puzzling*** algorithm implemented in TREE-PUZZLE as a heuristic strategy for a maximum likelihood tree search. PAUP* is also able to carry out a number of maximum-likelihood-based analyses on nucleotide sequence data. In particular, the program can perform exhaustive, branch-and-bound, and various types of heuristic searches, described previously, on aligned nucleotide sequences. The following example assumes that the HIV data set in NEXUS format (`hivALN.nex`) (see Box 8.4) has already been executed in PAUP*.

The first task in any maximum likelihood analysis is to choose an appropriate model. The default model in `PAUP*` is the HKY model (see Chapter 4) with a transition : transversion ratio of 2; however, this model should never be used uncritically. Although automated model-selection techniques are available and useful (Posada & Crandall, 1998) (see Chapter 10), we find that manual model selection can be highly informative and may identify good models that might otherwise be missed. Unfortunately, we cannot recommend a precise solution that is best for all situations (if such a solution were available, then it could simply be automated). Instead, we outline the following general strategy, which basically follows the "top-down dynamical" approach of Chapter 10, but without enforcing any particular *a priori* specification of models to be compared.

(1) Start with some reasonable tree for the data. This tree need not be optimal in any sense, but it should at least be a "good" tree under some criterion.
(2) Set the likelihood model to the most complex one available in PAUP* (six-substitution-type general time-reversible model with some fraction of invariable sites and rates at variable sites following a gamma distribution ($=$ GTR+ I+Γ)).
(3) Estimate all model parameters (i.e. relative substitution rates, base frequencies, proportion of invariable sites, gamma shape parameter) and calculate the likelihood score of the tree using the `LScores` command.

(4) Examine the parameter estimates carefully in an attempt to identify possible model simplifications that will not greatly reduce the fit of the model. Evaluate the simplified model and decide whether it is acceptable. If the attempted simplification is rejected, return to the previous model.
(5) Repeat Step 4 until no further acceptable simplifications can be found.
(6) Proceed to search for an optimal tree using the model and parameter estimates selected previously.

Some of these steps deserve further elaboration. The reason that the tree used for Step 1 is not critical is that parameter estimates do not vary much from tree to tree, as long as the trees are reasonably good explanations of the data (i.e. those that are much better than randomly chosen trees and include clades that are well supported under any optimality criterion) (Yang, 1994; Sullivan *et al.*, 1996; unpublished observations). The determination of whether to reject a simplified model in Step 4 can be accomplished using ***likelihood-ratio tests*** (LRTs), the ***AIC*** or ***BIC*** criteria (see Chapter 10), or more subjective methods. Model-selection criteria are somewhat controversial. LRTs offer the appeal of statistical rigor, but the choice of an appropriate alpha level for the test (e.g. 0.05, 0.01, 0.001) is arbitrary. Furthermore, even if a simple model is strongly rejected in favor of a more complex model, the simpler model may still be better for tree inference because it requires the estimation of fewer parameters. As models become more complex, more parameters must be estimated from the same amount of data, so that the variance of the estimates increases. Conversely, use of an oversimplified model generally leads to biased estimates. We follow Burnham and Anderson (1998) and others in preferring models that optimize the trade-off between bias and variance as a function of the number of parameters in the model. Unfortunately, it seems impossible to avoid some degree of subjectivity when selecting models, but this should not be taken as a reason to avoid model-based methods entirely (Sullivan & Swofford, 2001).

We illustrate our model-selection strategy using the HIV data set. First, we obtain a starting tree, which can be, for example, a parsimony, an NJ, or a maximum likelihood tree obtained under a simple model such as Jukes–Cantor. In practice, the choice of this tree rarely makes a difference in the model selected (unpublished observations). In the absence of further information, the LogDet transformation represents a good all-purpose distance, and an NJ tree computed from the LogDet distance matrix can serve as the starting tree:

```
dset distance=logdet; nj;
```

Steps 2 and 3 are then accomplished using the `LSet` and/or `LScores` commands:

```
lset nst=6 rmatrix=estimate basefreq=esti-
mate rates=gamma shape=estimate pinvar=estimate;
lscores;
```

Or, equivalently:

```
lscores/nst=6 rmatrix=estimate basefreq=esti-
mate rates=gamma shape=estimate pinvar=estimate;
```

NST represents the number of substitution types. Instead of two (i.e. transitions vs. transversions), it is assumed that all six of the pairwise substitution rates are potentially different. Γ-distributed rates for variable sites (see Chapter 4) are specified via `rates=gamma`. Rather than taking default values, all parameters are estimated from the data. The resulting output follows. Note that PAUP* reports the tree score as the negative log likelihood rather than the log likelihood itself; therefore, smaller likelihood scores are better. This is done so that all optimizations performed by the program can be treated as minimization problems.

```
Likelihood scores of tree(s) in memory:
  Likelihood settings:
    Number of substitution types = 6
    Substitution rate-matrix parameters estimated via ML
    Nucleotide frequencies estimated via ML
  Among-site rate variation:
  Assumed proportion of invariable sites = estimated
  Distribution of rates at variable sites = gamma
  (discrete approximation)
    Shape parameter (alpha) = estimated
    Number of rate categories = 4
    Representation of average rate for each category = mean
  These settings correspond to the GTR+G+I model
  Number of distinct data patterns under this model = 1013
  Molecular clock not enforced
  Starting branch lengths obtained using
    Rogers-Swofford approximation method
  Branch-length optimization = one-dimensional
    Newton-Raphson with pass limit = 20, delta = 1e-06
  -ln L (unconstrained) = 12463.69362

Tree                1
--------------------
-ln  L   17321.15334
Base frequencies:
  A         0.359090
  C         0.182946
  G         0.214583
  T         0.243381
```

```
Rate matrix R:
  AC        2.27643
  AG        4.89046
  AT        0.84752
  CG        1.14676
  CT        4.77744
  GT        1.00000
P_inv       0.159207
Shape       0.907088
```

Searching for simplifications, it is apparent that the substitution rates for the two transitions, r_{AG} and r_{CT}, are nearly equal (i.e. close to 4.8). Thus, an obvious first try at simplifying the model involves equating these two rates and reducing the number of parameters by one. This is accomplished by specifying a particular submodel of the GTR model using the `RClass` option, which allows entries in the rate matrix to be pooled into larger classes. The classification of substitution types follows the order r_{AC}, r_{AG}, r_{AT}, r_{CG}, r_{CT}, r_{GT}. All rates assigned the same alphabetic character are pooled into a single category. Thus, the following command pools the two transitions (r_{AG} and r_{CT}) into one class, but leaves each transversion distinct, and re-estimates all other parameters (all options from the previous command are still in effect):

```
lscores/rclass=(a b c d b e);
```

The resulting output shows that the tree score is only trivially worse than the full GTR+I+Γ model (17321.153 versus 17321.188):

```
Tree               1
--------------------
-ln L    17321.18757
Base frequencies:
  A         0.359680
  C         0.182335
  G         0.215248
  T         0.242737
Rate matrix R:
  AC        2.28603
  AG        4.84836
  AT        0.84841
  CG        1.14921
  CT        4.84836
  GT        1.00000
P_inv      0.159112
Shape      0.907157
```

Thus, it is appropriate to accept the simpler model with the two transitions pooled into one category.

Often, models with either Γ-distributed rates or invariable sites alone fit the data nearly as well as those with both types of rate variation. To examine this possibility for the HIV data set, the reduced model is evaluated, but with all sites variable (pinv = 0) and following a gamma distribution (still the current setting):

```
lscores/pinv=0;
```

The likelihood score obtained for this model is 17325.566, or 4.378 units worse. An LRT can be performed by multiplying this score difference by 2 and comparing to a chi-squared distribution with 1 degree of freedom (see Chapter 10). The *P*-value for this test is 0.0031, suggesting that the invariable sites plus gamma model (I + Γ) fits significantly better than the model alone. The AIC (see Chapter 10) also favors the I + Γ rate-variation model.

Next, we evaluate the model with some sites invariable, and all variable sites evolving at the same rate:

```
lsc/rates=equal pinv=est;
```

The likelihood score for this model, 17455.811, is even worse, so it also is rejected in favor of the I + Γ model.

After failing to simplify the model by eliminating among-site rate-variation parameters, attention can be returned to the substitution rate matrix. Inspection of this matrix reveals that the relative rate for the CG substitution ($r_{CG} = 1.149$) is not much different than $r_{GT} = 1$. Thus, these two substitutions also can be pooled and the resulting likelihood score calculated:

```
lsc/rclass=(a b c d b d) rates=gamma shape=estimate
pinv=estimate;
```

The resulting likelihood score and parameter estimates follow:

```
Tree                1
--------------------
-ln L    17321.52578
Base frequencies:
  A         0.359704
  C         0.183294
  G         0.215397
  T         0.241606
Rate matrix R:
  AC         2.16054
  AG         4.58543
  AT         0.80136
  CG        1.00000
  CT        4.58543
  GT        1.00000
P_inv      0.161272
Shape      0.914014
```

The likelihood score (17321.526) is not much worse than the previous acceptable model (17321.188). The difference in scores is equal to 0.338, and an LRT using twice this value with 1 degree of freedom yields a *P*-value of 0.41. Thus, the simpler model is preferred. Although we do not show the results, further attempts to simplify the model by additional substitution-pooling (e.g. `rclass=(a b c c b c)`) or assuming equal base frequencies (i.e. `basefreq=equal`) yield likelihood scores that are significantly worse than this model. It is interesting that the model chosen here, to our knowledge, has not been "named" and is not always available as a model setting in phylogenetic programs.

As an aside, the `RClass` option may be used to select other specific models. For example, `lset rclass=(a b a a c a);` can be used to specify the Tamura–Nei model (see Chapter 4), which assumes one rate for transversions, a second rate for purine transitions, and a third rate for pyrimidine transitions. `RClass` is meaningful only when the substitution rate matrix is being estimated `(rmatrix=estimate)` and the number of substitution types is set to six `(nst=6)`. Specific `(relative)` rates also may be specified using the `RMatrix` option. For example, the command

```
lset rmatrix=(1 5 1 1 2.5);
```

also specifies a Tamura–Nei model in which purine and pyrimidine transitions occur at a rate of 5 and 2.5 times (respectively) that of transversions. Note that these rates are assigned relative to the G–T rate, which is assigned a value of 1 and is not included in the list of `RMatrix` values.

Having chosen a model, we can now proceed to search for an optimal tree. To set the criterion to maximum likelihood, execute the following command:

```
set criterion=likelihood;
```

Because the criterion is now likelihood, the `HSearch` command will perform a maximum likelihood tree search using the default or current model settings. To reset the model chosen, the following command can be issued:

```
lset nst=6 rclass=(a b c d b d) rmatrix=estimate base-
freq=estimate rates=gamma shape=estimate pinv=estimate;
```

With these settings, PAUP* will estimate all model parameters on each tree evaluated in the search. Whereas in general this is the ideal method, it is too slow for data sets containing more than a small number of taxa. A more computationally tractable approach uses a successive approximations strategy that alternates between parameter estimation and tree search. First, the parameter values are estimated on a starting tree (e.g. the same NJ tree used for the model-selection procedure). These values are then fixed at their estimated values for the duration of a heuristic search. If the search finds a new tree that is better than the tree for which the parameter values were estimated, then the parameter values are re-estimated on this new tree. The iteration continues until the same tree is found in two consecutive searches. To implement this iterative strategy for the HIV example, the parameter values must first be fixed to the values estimated during the model-selection procedure. An easy way to do this is as follows (assuming that the NJ tree is still in memory from before):

```
lscores;
lset rmatrix=previous basefreq=previous shape=previous
pinv=previous;
```

Each parameter set to "`previous`" in this command is assigned the value that was most recently estimated for that parameter (i.e. in the preceding `LScores` command when the parameters were set to "`estimate`"). A standard heuristic search using simple stepwise addition followed by TBR rearrangements (see Section 8.4.2) is performed using the command `hseach`. If the search strategy options have been changed due to previous settings in this session, set them back to default (`AddSeq=simple NReps=10`):

```
hsearch;
```

A tree of likelihood score 17 316.909 is found, which is an improvement over the NJ starting tree. Consequently, the model parameters are re-estimated on this new tree:

```
lset rmatrix=estimate basefreq=estimate shape=estimate
pinv=estimate; Lscores;
```

With the newly optimized parameter values, the likelihood score of the tree found by the previous search improves only very slightly to 17316.896:

```
Tree                1
--------------------
-ln L    17316.89554
Base frequencies:
  A         0.359776
  C         0.183578
  G         0.215216
  T         0.241429
```

```
Rate matrix R:
  AC          2.17187
  AG          4.58751
  AT          0.80055
  CG          1.00000
  CT          4.58751
  GT          1.00000
P_inv       0.156966
Shape       0.900917
```

The parameter estimates are very close to those obtained using the original NJ tree (discussed previously), which is why the likelihood score did not improve appreciably. Nonetheless, to make sure that the change in parameter values will not lead to a new optimal tree, the search should be repeated, fixing the parameter values to these new estimates:

```
lset rmatrix=previous basefreq=previous shape=previous
pinv=previous;
hsearch;
```

This search finds the same tree as the previous one did, so the iterations can stop. Analyses of both real and simulated data sets provide reassurance that this iterative, successive-approximations strategy can be safely used as a much faster approach in ML tree topology estimation (Sullivan *et al.*, 2005).

Although it is not important for the relatively small HIV data set, it may be desirable for data sets containing larger numbers of taxa to reduce the scope of the rearrangements using the `ReconLimit` option. If `ReconLimit` is set to *x*, then any rearrangement for which the reconnection distance (see Section 8.4.2) exceeds *x* is not evaluated. For data sets of 30 or more taxa, use of values such as `ReconLimit=12` can often substantially reduce the time required for a TBR search without greatly compromising its effectiveness.

As a starting tree for the heuristic search, it is possible to use an NJ tree (instead of a tree obtained by stepwise addition) with the command `HSearch Start = NJ`;. Although the program finds the same maximum likelihood tree for the HIV example data set, this will not necessarily be true for other data sets. In general, the use of different starting trees permits a more thorough exploration of the tree space, and it is always a good idea to compare results of alternative search strategies. Use of random-addition sequences (i.e. `HSearch AddSeq=Random`;) is especially desirable. The default number of random-addition-sequence replicates is rather small (10); the option `NReps=`*n* can be used to specify a different number of replicates *n*.

As for parsimony and distance methods, PAUP* can perform bootstrapping and jackknifing analyses under the likelihood criterion, although these analyses may be very time-consuming. In addition, a zero-branch-length test can be performed on the maximum likelihood tree by setting the `ZeroLenTest` option of the `LSet` command to full and then executing the `DescribeTrees` command:

```
lset zerolentest=full; describetrees;
```

If more than one tree is in memory, the result of the test is printed on the screen (and/or in the log file: see Box 8.4) for each tree. In the zero-branch-length test, the statistical support for each branch of a previously estimated maximum likelihood tree is obtained as follows. After collapsing the branch under consideration (by constraining its length to zero), the likelihood of the tree is recalculated and compared with the likelihood of the original tree. If the likelihood of the former is significantly worse than that of the latter according to an LRT, the branch is considered statistically supported. As a result of executing these commands, PAUP* prints out a list of all pairs of neighboring nodes in the tree (including terminal nodes), each pair with a corresponding *p-value* giving the support for the branch connecting the two nodes:

```
Branch lengths and linkages for tree #1 (unrooted)

Node          Connected      Branch      Standard      --- L.R. test ---
               to node       length        error        lnL diff      P*
--------------------------------------------------------------------------
L20571(1)         26         0.95536     0.06838        399.948   <0.001
    15            26         0.10463     0.02367          8.449   <0.001
AF103818 (2)      15         0.33927     0.02887        137.486   <0.001
X52154 (3)        15         0.33452     0.02858        125.842   <0.001
    25            26         0.21406     0.02630         32.721   <0.001
U09127 (4)        25         0.07713     0.00837         46.112   <0.001
    24            25         0.02213     0.00682          4.197    0.004
    16            24         0.06190     0.00724        110.645   <0.001
U27426 (5)        16         0.06438     0.00663        132.276   <0.001
U27445 (6)        16         0.04666     0.00582         86.951   <0.001
    23            24         0.01643     0.00438         14.143   <0.001
    17            23         0.07067     0.00754        157.899   <0.001
AF067158 (7)      17         0.05477     0.00603        126.407   <0.001
U09126 (8)        17         0.04146     0.00536         73.886   <0.001
    22            23         0.03517     0.00583         47.726   <0.001
    18            22         0.04925     0.00614         99.613   <0.001
U27399 (9)        18         0.04582     0.00557         99.612   <0.001
U43386 (10)       18         0.05442     0.00601        132.606   <0.001
    21            22         0.03817     0.00549         69.776   <0.001
    20            21         0.01060     0.00319         11.560   <0.001
    19            20         0.00639     0.00237          8.840   <0.001
L02317 (11)       19         0.03638     0.00456        136.741   <0.001
```

```
AF042106 (14)     19          0.04899     0.00529        199.615   <0.001
AF025763 (12)     20          0.04503     0.00512        190.207   <0.001
U08443 (13)       21          0.04405     0.00519        138.658   <0.001
---------------------------------------------------------------------------
Sum                           2.81762

* Probability of obtaining a likelihood ratio as large or larger than the
observed ratio under the null
hypothesis that a branch has zero length (full reoptimization after forc-
ing a branch length to zero)

-Ln likelihood = 17316.89554
```

In this case, all the branches in the maximum likelihood tree seem to be robustly supported by this test. However, the test only assesses whether a given resolution of a trichotomy is a significant improvement relative to leaving the trichotomy unresolved. For example, it is possible for incompatible groups on different trees to both receive significant support using this test. The fact that not all of the clades received high bootstrap values in the parsimony and NJ trees, as well as in maximum likelihood bootstrap analyses (not shown), highlights the need for caution in interpreting the results of this test.