7

Building phylogenetic trees

In the previous chapter, we considered the problem of multiple alignment of sets of sequences. One can argue [Sankoff, Morel & Cedergren 1973] that alignment of sequences should take account of their evolutionary relationship. For example, an alignment that implies many substitutions between closely related sequences is less plausible than one that makes most of its changes over large evolutionary distances.

Some multiple alignment algorithms use a tree; for instance, we have seen that several progressive alignment algorithms use a 'guide tree'. As the name suggests, this tree is meant to guide the clustering process rather than satisfy a taxonomist. In this chapter we shift emphasis, and begin to take a serious interest in building trees. However, we do not lose sight of alignment: the last section describes methods for simultaneous alignment and tree building.

We concentrate here on two general approaches to tree building: distance methods and parsimony; the next chapter formulates phylogeny probabilistically.

7.1 The tree of life

The similarity of molecular mechanisms of the organisms that have been studied strongly suggests that all organisms on Earth had a common ancestor. Thus any set of species is related, and this relationship is called a *phylogeny*. Usually the relationship can be represented by a *phylogenetic tree*. The task of phylogenetics is to infer this tree from observations upon the existing organisms.

Traditionally, morphological characters (both from living and fossilised organisms) have been used for inferring phylogenies. Zuckerkandl & Pauling's pioneering paper [1962] showed that molecular sequences provide sets of characters that can carry a large amount of information. If we have a set of sequences from different species, therefore, we may be able to use them to infer a likely phylogeny of the species in question. This assumes that the sequences have descended from some common ancestral gene in a common ancestral species.

The widespread occurrence of gene duplication means that the foregoing assumption needs to be checked carefully. The phylogenetic tree of a group of sequences does not necessarily reflect the phylogenetic tree of their host species,

because gene duplication is another mechanism, in addition to speciation, by which two sequences can be separated and diverge from a common ancestor. Genes which diverged because of speciation are called *orthologues*. Genes which diverged by gene duplication are called *paralogues*. If we are interested in inferring the phylogenetic tree of the species carrying the genes, we must use orthologous sequences. But, of course, we might be interested in the phylogeny of duplication events, in which case we might construct a phylogeny of paralogues, even the paralogues within a single species. The distinction between paralogues and orthologues is illustrated by Figure 7.1.

7.2 Background on trees

In this chapter, all trees will be assumed to be binary, meaning that an edge that branches splits into two daughter edges (Figure 7.2). This is equivalent to saying that three edges meet at every branch node, a *node* being an endpoint of an edge. The assumption that the tree is binary is not a serious limitation, because any other branching pattern can be approximated by a binary tree in which some of the branches are very short.

Each edge of the tree has a certain amount of evolutionary divergence associated to it, defined by some measure of distance between sequences, or from a model of substitution of residues over the course of evolution. We adopt the general term 'length' or 'edge length' here, and represent this by the lengths of edges in the figures we draw. The relationship between phylogenetically determined lengths and palaeontological time periods was examined by Langley & Fitch [1974], who found that different proteins can change at very different rates, and the same sequence can evolve much faster in some organisms than others. However, averaging over larger sets of proteins does demonstrate a broad correspondence between lengths and evolutionary time periods [Doolittle *et al.* 1996; Wray, Levinto & Shapiro 1996].

A true biological phylogeny has a 'root', or ultimate ancestor of all the sequences. Some algorithms provide information, or at least a conjecture, about the location of the root. Others, like parsimony and the probabilistic models in the next chapter, are completely uninformative about its position, and other criteria have to be used for rooting the tree. We consider here how to represent both rooted and unrooted trees.

Figure 7.2 shows an unrooted tree and a rooted version of it. Note that, in the latter, we have drawn the root at the top, with the *leaves*, the terminal nodes corresponding to the observed sequences, at the base.

The leaves of trees have names or numbers. Sometimes these can be swopped without altering the phylogeny (e.g. numbers 4 and 5 in Figure 7.2), but they often cannot (e.g. swopping 1 and 2 in the figure changes the phylogeny). A tree

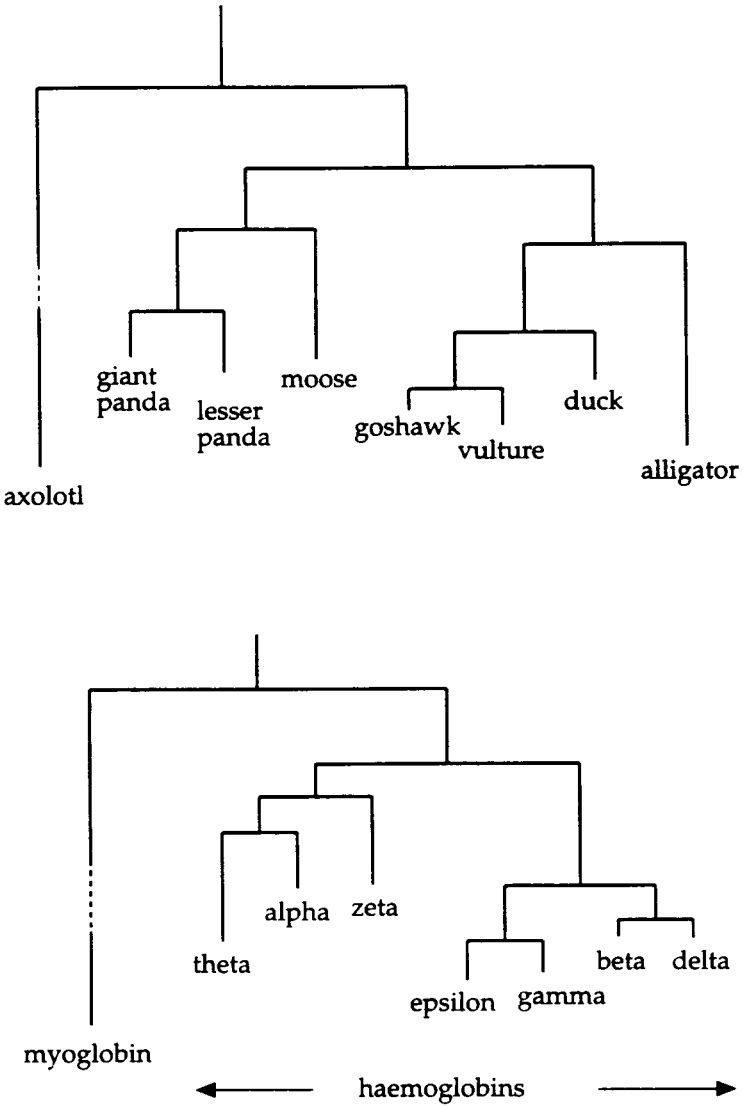

Figure 7.1 *Above: a tree of orthologues based on a set of alpha haemoglobins. Below: a tree of paralogues, the alpha, beta, gamma, delta, epsilon, zeta and theta chains of human haemoglobins, and human myoglobin. The orthologues are the alpha haemoglobins with* SWISS-PROT *identifiers* HBA_ACCGE, HBA_AEGMO, HBA_AILFU, HBA_AILME, HBA_ALCAA, HBA_ALLMI, HBA_AMBME, *and* HBA_ANAPL, *chosen because they were alphabetically the first eight alpha globins in* PFAM *[Sonnhammer, Eddy & Durbin 1997] (http://genome.wustl.edu/Pfam/). The paralogues are globins with* SWISS-PROT *identifiers* HBAT_HUMAN, HBAZ_HUMAN, HBA_HUMAN, HBB_HUMAN, HBD_HUMAN, HBE_HUMAN, HBG_HUMAN, *and* MYG_HUMAN. *The trees were made by neighbour-joining, Section 7.3, using J. Felsenstein's package* PHYLIP *(http://evolution.genetics.washington.edu/phylip.html). The distances used for neighbour-joining were the* PAM*-based ML distances (see p. 228) determined by the program* PROTDIST *in* PHYLIP.

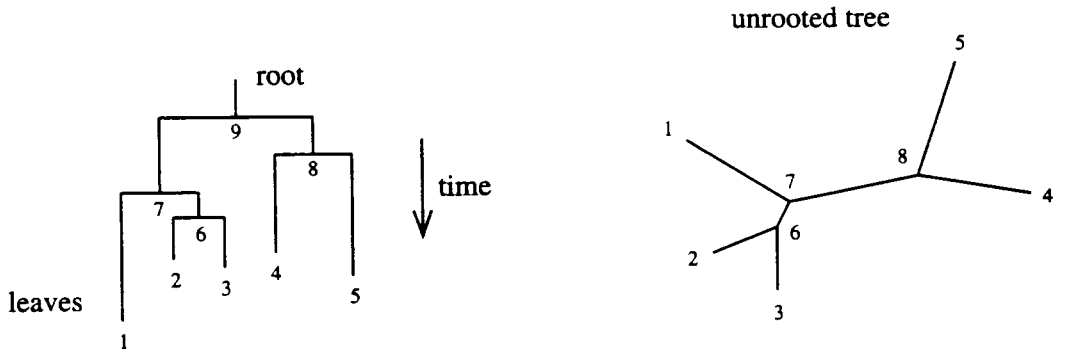

Figure 7.2 *An example of a binary tree, showing the root and leaves, and the direction of evolutionary time (the most recent time being at the bottom of the figure). The corresponding unrooted tree is also shown; the direction of time here is undetermined.*

with a given labelling will be called a *labelled branching pattern*. More loosely, we refer to this as the tree *topology*[1] and denote it by the symbol T. To complete the definition of a phylogenetic tree, one must also define the lengths of its edges; these will generally be denoted[2] by t_i with a suitable numbering scheme for the is.

Counting and labelling trees

The nodes and edges of a rooted tree can be counted as follows: Suppose there are n leaves. As we move up the tree, the edges coalesce as each new node is reached. Each time this happens, the number of edges is reduced by one. So there must be $(n-1)$ nodes in addition to the n leaves, giving $(2n-1)$ nodes in all, and one fewer edges, i.e. $(2n-2)$, discounting the edge above the root node. We shall label the leaves using the numbers 1 to n, and assign the branch nodes the numbers $n+1$ to $2n-1$, reserving $2n-1$ for the root node. The lengths of edges will be labelled by the node at the bottom of the edge, so d_1 is the length associated to the edge above node 1, and so on.

An unrooted tree with n leaves has $2n-2$ nodes altogether and $2n-3$ edges. A root can be added to it at any of its edges, thereby producing $(2n-3)$ rooted trees from it. Figure 7.3 shows this for $n=3$; the three positions for the root yield three rooted trees. There are therefore $(2n-3)$ times as many rooted trees as unrooted trees, for a given number n of leaves.

Instead of the root, we can add an extra edge or 'branch' with a distinct label at its leaf (i.e. a '4') to the unrooted tree with three leaves in Figure 7.3, thereby obtaining an unrooted tree with four leaves. There are three such trees, with

[1] A topologist would reserve this term for the unlabelled branching patterns, i.e. the distinct classes of tree that cannot be rearranged into each other by permutation of edges at nodes or shrinking or extending of edges.

[2] A deliberate echo of 'time', the variable we are ultimately interested in.

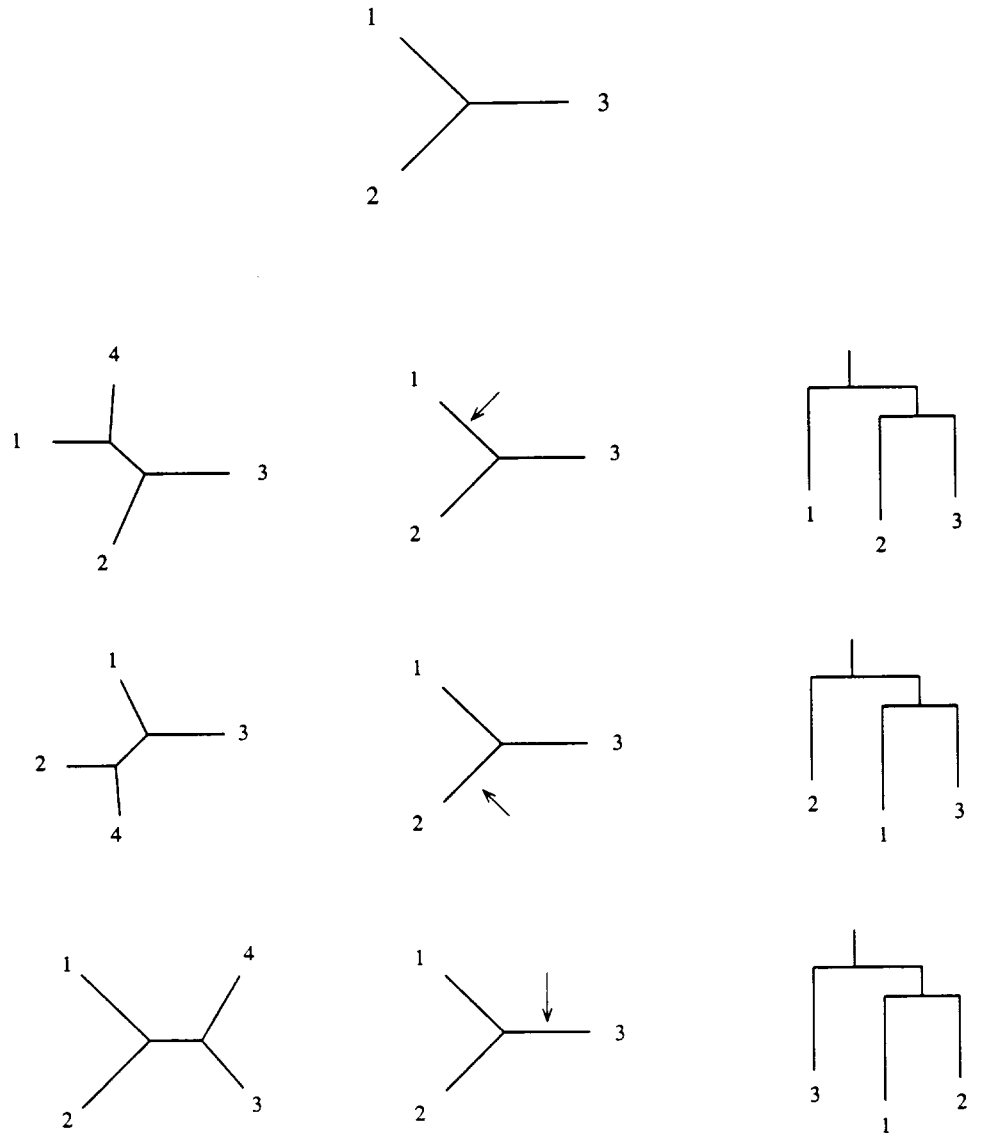

Figure 7.3 *The rooted trees (right-hand column) derived from the unrooted tree for three sequences by picking different edges as positions for the root (arrows).*

$(2n-3) = 5$ edges, and it is easy to see that they are distinct labelled branching patterns. There are then five ways of adding a further branch labelled with a distinct label ('5'), giving in all $3 \times 5 = 15$ unrooted trees with five leaves. Continuing this, we see that there are $(3) \cdot (5) \cdot \ldots \cdot (2n-5)$ unrooted trees with n leaves; this number is also written $(2n-5)!!$. From what was said above, it follows that there are $(2n-3)!!$ rooted trees. The number of trees grows very rapidly with n; for $n = 10$ there are about two million unrooted trees, and for $n = 20$, 2.2×10^{20} of them. For further information on tree counting, see Felsenstein [1978b].

Exercises

7.1 Draw the rooted trees obtained by adding the root in all seven possible positions to the unrooted tree in Figure 7.2.

7.2 The trees with three and four leaves in Figure 7.3 all have the same unlabelled branching pattern. For both rooted and unrooted trees, how many leaves do there have to be to obtain more than one unlabelled branching pattern? Find a recurrence relation for the number of rooted trees. (Hint: Consider the trees formed by joining two trees at their roots.)

7.3 All trees considered so far have been binary, but one can envisage ternary trees that, in their rooted form, have *three* branches descending from a branch node. The unrooted trees therefore have four edges radiating from every branch node. If there are m branch nodes in an unrooted ternary tree, how many leaves are there and how many edges?

7.4 Consider next a composite unrooted tree with m ternary branch nodes and n binary branch nodes. How many leaves are there, and how many edges? Let $N_{m,n}$ denote the number of distinct labelled branching patterns of this tree. Extend the counting argument for binary trees to show that

$$N_{m,n} = (3m + 2n - 1)N_{m,n-1} + (n+1)N_{m-1,n+1}$$

(Hint: the first term after the '=' counts the number of ways that a new edge can be added to an existing edge, thereby creating an additional binary node; the second term corresponds to edges added at binary nodes, thereby producing ternary nodes.)

7.5 Use the above recurrence relationship to calculate $N_{m,0}$, the number of distinct pure ternary trees with m branch nodes, for small values of m. (Hint: We know that $N_{0,i} = (2i-1)!!$, and the recurrence relationship allows one to express $N_{m,0}$ in terms of $N_{0,i}$, for $i \leq n$. Programmers will enjoy writing a recursive program that carries out this operation.) Check that the calculated numbers satisfy $N_{m,0} = \prod_{i=1}^{m}(1+9i(i-1)/2)$. Can you prove this formula?

7.3 Making a tree from pairwise distances

Some of the more intuitively accessible methods of tree building begin with a set of distances d_{ij} between each pair i, j of sequences in the given dataset. There are many different ways of defining distance. For example, one can take d_{ij} to be the fraction f of sites u where residues x_u^i and x_u^j differ (presupposing an alignment of the two sequences). This gives a sensible definition for small fractions f. For two unrelated sequences, however, random substitutions will cause f to approach the fraction of differences expected by chance, and we would like the distance

to become large as f tends to this value. Markov models of residue substitution, such as the Jukes–Cantor model for DNA (p. 195), can be used to define distances that behave this way. Thus the Jukes–Cantor distance is $d_{ij} = -\frac{3}{4}\log(1-4f/3)$, which tends to infinity as the equilibrium value of f (75% of residues different) is approached. We return to the definition of distances in Section 8.6.

Clustering methods: UPGMA

We begin with a clustering procedure [Sokal & Michener 1958] called UPGMA, which stands for unweighted pair group method using arithmetic averages. Despite its formidable acronym, the method is simple and intuitively appealing. It works by clustering the sequences, at each stage amalgamating two clusters and at the same time creating a new node on a tree. The tree can be imagined as being assembled upwards, each node being added above the others, and the edge lengths being determined by the difference in the heights of the nodes at the top and bottom of an edge.

First we define the distance d_{ij} between two clusters C_i and C_j to be the average distance between pairs of sequences from each cluster:

$$d_{ij} = \frac{1}{|C_i||C_j|} \sum_{p \text{ in } C_i, q \text{ in } C_j} d_{pq}, \tag{7.1}$$

where $|C_i|$ and $|C_j|$ denote the number of sequences in clusters i and j, respectively. Note that, if C_k is the union of the two clusters C_i and C_j, i.e. if $C_k = C_i \cup C_j$, and if C_l is any other cluster, then (Exercise 7.6):

$$d_{kl} = \frac{d_{il}|C_i| + d_{jl}|C_j|}{|C_i| + |C_j|}. \tag{7.2}$$

The clustering procedure is:

Algorithm: UPGMA

Initialisation:

Assign each sequence i to its own cluster C_i,

Define one leaf of T for each sequence, and place at height zero.

Iteration:

Determine the two clusters i, j for which d_{ij} is minimal. (If there are several equidistant minimal pairs, pick one randomly.)

Define a new cluster k by $C_k = C_i \cup C_j$, and define d_{kl} for all l by (7.2).

Define a node k with daughter nodes i and j, and place it at height $d_{ij}/2$.

Add k to the current clusters and remove i and j.

Termination:

When only two clusters i, j remain, place the root at height $d_{ij}/2$. ◁

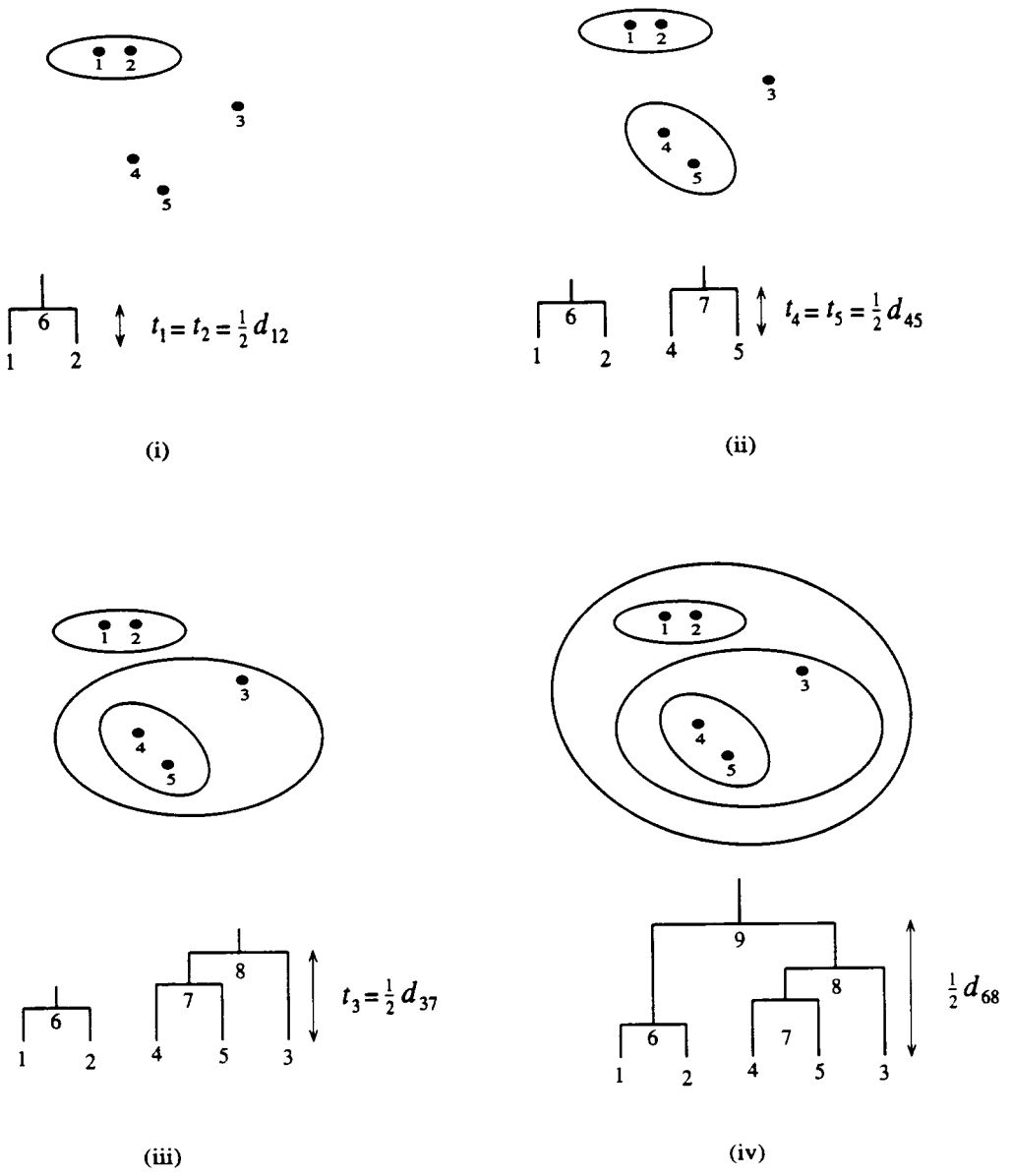

Figure 7.4 *An example of how UPGMA produces a rooted tree by successively clustering sequences, in this case a set of five sequences whose distances can be represented by points in the plane (this will not generally be true of a set of distances).*

To check that this procedure produces well-defined edge lengths, we have to show that a parent node always lies above its daughters (see Exercise 7.7). There are variants of UPGMA that define the distance between clusters as the minimum or maximum of the distances between constituent sequences, rather than the average, but UPGMA seems to have the best performance record.

Example: UPGMA applied to five sequences

The distances between five sequences are represented schematically as distances in the plane (Figure 7.4). UPGMA works as follows: First, the two closest sequences are found; suppose these are x^1 and x^2. Their parent node is given the number 6 and edge lengths t_1 and t_2 defined by $t_1 = t_2 = \frac{1}{2}d_{12}$. Next, we define the distance d_{i6} between a sequence x^i and the new branch node 6, representing the cluster $\{x^1, x^2\}$, to be the average $\frac{1}{2}(d_{1i} + d_{2i})$, and search for the closest pair amongst all remaining sequences and node 6. This pair is $\{x^4, x^5\}$; their parent node, node 7, is constructed as above and edge lengths t_4 and t_5 defined by $d_4 = d_5 = \frac{1}{2}d_{45}$. This process is repeated. The next closest pair is x^3 and node 7. A parent node, node 8, to x^3 and node 7 is introduced, and the edge above x^3 assigned a length $t_3 = \frac{1}{2}d_{37}$, and the edge above node 7 a length $t_7 = \frac{1}{2}d_{37} - \frac{1}{2}d_{45}$, so that the sum of times down all branches is the same. The last amalgamation occurs between node 6 ($\{x^1, x^2\}$) and node 8 ($\{x^3, x^4, x^5\}$), with a distance $d_{68} = \frac{1}{6}(d_{13} + d_{14} + d_{15} + d_{23} + d_{24} + d_{25})$. □

Exercises

7.6 Show that, if distances between clusters are defined by (7.1), and if $C_k = C_i \cup C_j$, then d_{kl} for any l is given by (7.2).

7.7 Show that a node always lies above its daughter nodes. (Hint: if not, show that an incorrect choice of closest clusters would have been made when one of the daughters was formed.)

Molecular clocks and the ultrametric property of distances

UPGMA produces a rooted tree of a special kind. The edge lengths in the resulting tree can be viewed as times measured by a *molecular clock* with a constant rate. The divergence of sequences is assumed to occur at the same constant rate at all points in the tree, which is equivalent to saying that the sum of times down a path to the leaves from any node is the same, whatever the choice of path. If our distance data are derived by adding up edge lengths in a tree T with a molecular clock, then UPGMA will reconstruct T correctly. To see this, imagine a horizontal line rising through the tree T starting from the level of the leaves: each time it crosses a node, the distances of all the leaves in the left branch from that node to the leaves in the right branch will be the current minimum distance, and a node will therefore be added precisely where the node is encountered in the original tree T.

If the original tree is not well-behaved in this way, but has different length routes to its leaves, as in Figure 7.5 (left), then it may be reconstructed incorrectly by UPGMA (Figure 7.5 right). What goes wrong in this case is that the closest leaves are not neighbouring leaves: they do not have a common parent node. A test of whether reconstruction is likely to be correct is the *ultrametric*

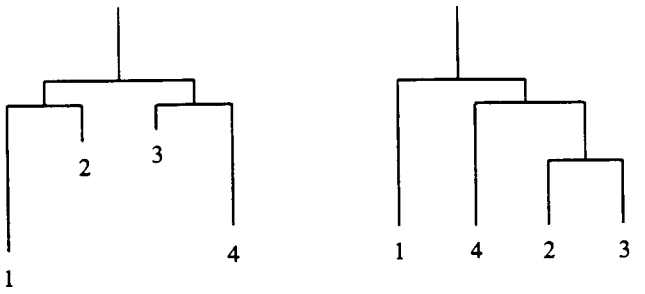

Figure 7.5 *A tree (left) that is reconstructed incorrectly by UPGMA (right).*

condition. The distances d_{ij} are said to be ultrametric if, for any triplet of sequences, x^i, x^j, x^k, the distances d_{ij}, d_{jk}, d_{ik} are either all equal, or two are equal and the remaining one is smaller. This condition holds for distances derived from a tree with a molecular clock.

Exercise

7.8 It can be shown that, if the distances d_{ij} are ultrametric, and if a tree is constructed from these distances by UPGMA, then the distances obtained from this tree by taking twice the height of the node on the path between i and j are identical to the d_{ij}. Check that this is true in the example of UPGMA applied to five sequences if the distances are ultrametric. (Hint: Show that, when two clusters C_k and C_l are amalgamated, the ultrametric condition implies that the distances between any leaf in C_k and any leaf in C_l are the same.)

Additivity and neighbour-joining

In describing the molecular clock property of the trees produced by UPGMA, we implicitly assumed another important property: additivity. Given a tree, its edge lengths are said to be *additive* if the distance between any pair of leaves is the sum of the lengths of the edges on the path connecting them. This property is built in automatically as the UPGMA tree is constructed. However, it is possible for the molecular clock property to fail but for additivity to hold, and in that case there are algorithms that can be used to reconstruct the tree correctly.

Given a tree T with additive lengths $\{d_\bullet\}$, we can try to reconstruct it from the pairwise distances of its leaves $\{d_{ij}\}$ as follows: Find a pair of *neighbouring leaves*, i.e. leaves that have the same parent node, k. Suppose their numbers are i, j. Remove them from the list of leaf nodes and add k to the current list of nodes, defining its distance to leaf m by

$$d_{km} = \tfrac{1}{2}(d_{im} + d_{jm} - d_{ij}). \tag{7.3}$$

By additivity, the distances d_{km} just defined are precisely those between the equiv-

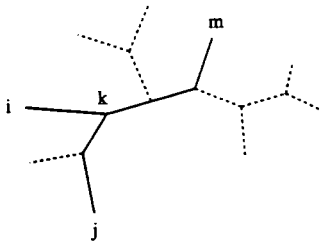

Figure 7.6 *For any three leaves i, j and m there is a node, k here, where the branches to them meet. By additivity, $d_{im} = d_{ik} + d_{km}$, $d_{jm} = d_{jk} + d_{km}$ and $d_{ij} = d_{ik} + d_{jk}$, from which it follows that $d_{km} = \frac{1}{2}(d_{im} + d_{jm} - d_{ij})$, which is equation (7.3).*

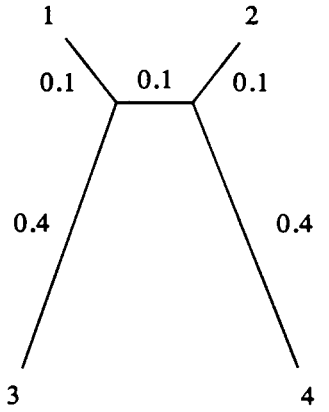

Figure 7.7 *A tree whose closest pair of leaves are not neighbouring leaves. The lengths of edges are shown. If the lengths are additive, we find $d_{12} = 0.3$ and $d_{13} = 0.5$, so the neighbouring pair 1,3 are further apart than the non-neighbouring pair 1,2.*

alent nodes in the original tree (see Figure 7.6). In this way we can strip away leaves, reducing the number by one at each operation, until we get down to a pair of leaves.

If we could determine from distances alone a pair of neighbouring leaves, therefore, we could reconstruct a tree with additive lengths exactly. The remarkable fact is that we can pick neighbouring leaves, using a procedure proposed by Saitou & Nei [1987] and modified by Studier & Keppler [1988].

First, note that it does *not* suffice to pick simply the two closest leaves, i.e. the pair i,j with d_{ij} minimal. Figure 7.7 shows why. If one of a pair of neighbours has a short edge and the other a long edge, the one with the short edge may be closer to another leaf than its true neighbour, as happens in the illustrated tree. To avoid this, the trick is to subtract the averaged distances to all other leaves; in effect, this compensates for long edges. We define

$$D_{ij} = d_{ij} - (r_i + r_j),$$

where

$$r_i = \frac{1}{|L|-2}\sum_{k\in L} d_{ik}, \tag{7.4}$$

and $|L|$ denotes the size of the set L of leaves. The claim now is that a pair of leaves i, j for which D_{ij} is minimal will be neighbouring leaves; we give a proof at the end of this chapter. It is instructive to check that this is true of the tree in Figure 7.7 (see Exercise 7.9).

The complete algorithm for neighbour-joining works by constructing a tree T by steps, keeping a list L of active nodes in this tree. If there were a pre-existing additive tree, L would be the current remaining set of leaf nodes as neighbouring pairs were stripped away, and T would be the tree built up from these stripped-off nodes.

Algorithm: Neighbour-joining

Initialisation:

Define T to be the set of leaf nodes, one for each given sequence, and put $L = T$.

Iteration:

Pick a pair i, j in L for which D_{ij}, defined by (7.4), is minimal.

Define a new node k and set $d_{km} = \frac{1}{2}(d_{im} + d_{jm} - d_{ij})$, for all m in L.

Add k to T with edges of lengths $d_{ik} = \frac{1}{2}(d_{ij} + r_i - r_j)$, $d_{jk} = d_{ij} - d_{ik}$, joining k to i and j, respectively.

Remove i and j from L and add k.

Termination:

When L consists of two leaves i and j add the remaining edge between i and j, with length d_{ij}.

◁

The definition of the length d_{ik} by $\frac{1}{2}(d_{ij} + r_i - r_j)$ gives the correct length if additivity holds, since this expression is the average of $\frac{1}{2}(d_{ij} + d_{im} - d_{jm})$ over all leaves m, and each such term is just d_{ik} (compare (7.3)).

Additivity is a property that depends on the distance measure used: a tree may be additive with respect to one distance measure and not with respect to another. In Section 8.6 we shall see that a certain type of maximum likelihood distance measure would be expected to give additivity, in the limit of a large amount of data, if the underlying model assumptions were correct. Real data, of course, will only be at best approximately additive.

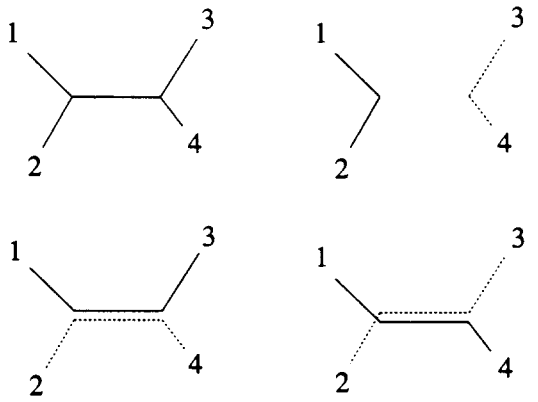

Figure 7.8 *Additivity means that two of the summed lengths $d_{12}+d_{34}$, $d_{13}+d_{24}$, $d_{14}+d_{23}$ must be larger than the third and equal in size. This holds if the pairwise distances are obtained by summing edge lengths, as the diagrams show.*

We can use neighbour-joining even if lengths are not additive, but reconstruction of the correct tree is no longer guaranteed. Just as the ultrametric condition provided a test for the molecular clock property, so we can use the following property of distances as a test for additivity: For every set of four leaves i, j, k and l, two of the distances $d_{ij}+d_{kl}$, $d_{ik}+d_{jl}$ and $d_{il}+d_{jk}$ must be equal and larger than the third. This *four-point condition* is a consequence of additivity, because two of the sums include the length of the 'bridge' connecting pairs of leaves (see Figure 7.8).

Exercises

7.9 Show that the smallest distances D_{ij} in the tree in Figure 7.7 correspond to neighbouring leaves.

7.10 Show that, for a tree with four leaves, D_{ij} for a pair of neighbours is less than D_{ij} for all other pairs by twice the 'bridge length', i.e. the length of the edge joining the two branch nodes in the tree.

Rooting trees

Neighbour-joining, unlike UPGMA, produces unrooted trees. Finding the root is a secondary task, which can be accomplished by adding an *outgroup*, or species that is known to be more distantly related to each of the remaining species than they are to each other. The point in the tree where the edge to the outgroup joins is therefore the best candidate for the root position. In the top tree in Figure 7.1, for instance, the axolotl can be treated as an outgroup, as it is an amphibian whereas all the other species are amniotes. It is therefore reasonable to place the divide between the axolotl and the other species earlier than any of the other branches.

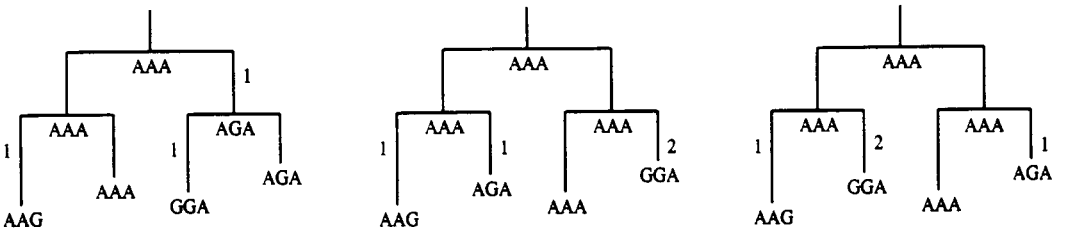

Figure 7.9 *Building a tree by parsimony.*

In the absence of a convenient outgroup, there are somewhat *ad hoc* strategies, such as picking the midpoint of the longest chain of consecutive edges, which would be expected to identify the root if deviations from a molecular clock were not too great.

7.4 Parsimony

We come now to what is probably the most widely used of all tree building algorithms: *parsimony*. It works by finding the tree which can explain the observed sequences with a minimal number of substitutions. It uses a different general strategy from the distance-based algorithms considered so far. Instead of building a tree, it assigns a cost to a given tree, and it is necessary to search through all topologies, or to pursue a more efficient search strategy that achieves this effect (see p. 176), in order to identify the 'best' tree. We can therefore distinguish two components to the algorithm:

(1) the computation of a cost for a given tree T;

(2) a search through all trees, to find the overall minimum of this cost.

We begin with an example. Suppose we have the following four aligned nucleotide sequences:

AAG
AAA
GGA
AGA

We can try out different trees for these four sequences and count the number of substitutions needed in each tree, summing over all sites. Figure 7.9 shows three possible trees for the above four sequences; they differ in the order in which the sequences are assigned to the leaves. In each tree, hypothetical sequences have been assigned to the ancestral nodes so as to minimise the number of changes needed in the whole tree. We shall see shortly how this is done. The leftmost tree needs fewer changes (a total of three) than the two shown to its right (which need four each).

As we see in this example, parsimony treats each site independently, and then

adds the substitutions for all sites. The basic step is therefore counting the minimal number of changes that need to be made at one site, given a topology and an assignment of residues to the leaves. There is a simple algorithm to carry out this step. Consider first a slight extension of parsimony, called *weighted parsimony*, that doesn't just count the number of substitutions, but adds costs $S(a,b)$ for each substitution of a by b; the aim is now to minimise this cost [Sankoff & Cedergren 1983]. Weighted parsimony reduces to traditional parsimony when $S(a,a) = 0$ for all a, and $S(a,b) = 1$ for all $a \neq b$.

To compute the minimal cost at site u we proceed as follows: Let $S_k(a)$ denote the minimal cost for the assignment of a to node k.

Algorithm: Weighted parsimony

Initialisation:

Set $k = 2n - 1$, the number of the root node.

Recursion: Compute $S_k(a)$ for all a as follows:

If k is leaf node:

Set $S_k(a) = 0$ for $a = x_u^k$, $S_k(a) = \infty$, otherwise.

If k is not a leaf node:

Compute $S_i(a)$, $S_j(a)$ for all a at the daughter nodes i, j, and define $S_k(a) = \min_b(S_i(b) + S(a,b)) + \min_b(S_j(b) + S(a,b))$.

Termination:

Minimal cost of tree $= \min_a S_{2n-1}(a)$.

◁

Note that the steps under 'Recursion' require that S_i and S_j are computed for the daughter nodes i, j of k, and this is achieved by returning to 'Recursion' for both i and j. The effect is that the algorithm starts at the leaves and works its way up to the root. This way of passing through a tree is called *post-order traversal*, and plays an important part in many computer implementations of tree algorithms.

It is sometimes of interest to find the ancestral assignments of residues that give the minimal cost. For instance, one way of defining a length for an edge is to count the number of mismatches along that edge that occur in all possible minimal-cost ancestral assignments to the tree. This can be achieved by keeping pointers from each residue a at node k to those residues b and c at daughter nodes i and j, respectively, that were the minimising choices in the equation defining $S_k(a)$ in the weighted parsimony algorithm. We define pointers $l_k(a)$, $r_k(a)$ to left and right daughters of node k, respectively (these pointers perhaps having more than one target if there are several possible minimising residues), and add the

steps

$$\text{Set } l_k(a) = \text{argmin}_b(S_i(b) + S(a,b)),$$
$$\text{and } r_k(a) = \text{argmin}_b(S_j(b) + S(a,b)). \tag{7.5}$$

at the end of the 'Recursion' block of the weighted parsimony algorithm. To obtain an assignment of ancestral residues, we pick a residue a at the root that gives the minimal cost $S_{2n-1}(a)$ and trace back to the leaves using the pointers, choosing arbitrarily whenever the pointers have several possible targets.

In the case of traditional parsimony, where we just count the number of substitutions, all that is needed to obtain the cost of the tree is to keep a list of minimal cost residues at each node, together with the current cost C.

Algorithm: Traditional parsimony [Fitch 1971]

Initialization:

Set $C = 0$ and $k = 2n - 1$.

Recursion: To obtain the set R_k:

If k is leaf node:

Set $R_k = x_u^k$.

If k is not a leaf node:

Compute R_i, R_j for the daughter nodes i, j of k, and set

$R_k = R_i \cap R_j$ if this intersection is not empty, or else

set $R_k = R_i \cup R_j$ and increment C.

Termination:

Minimal cost of tree$= C$.

◁

There is a traceback procedure for finding ancestral assignments in traditional parsimony: We choose a residue from R_{2n-1}, then proceed down the tree. Having chosen a residue from the set R_k, we pick the same residue from the daughter set R_i if possible, and otherwise pick a residue at random from R_i (and similarly for the other daughter set R_j).

The tree T in Figure 7.10 shows two possible sets of ancestral residues obtained by this traceback procedure (the two middle figures). The bottom left figure shows another assignment that cannot be obtained this way. The reason for this failure is that keeping a list of minimal cost residues at each node neglects the possibility that a mismatch cost can be paid at some level in the tree and recouped higher up. This is automatically taken care of with the algorithm for weighted parsimony. The bottom right figure shows the minimal costs for A, B at each node, and the particular choices of back-pointers defined by (7.5) that lead

to the assignments in the tree on the bottom left. Note that the left pointer from the top node goes to the residue B whose cost is one more than the minimum. The difference of one is recouped because a B does not have to pay a mismatch penalty for this transition. In general, the assignments not obtained by traceback with R_k can be found by keeping a set Q_k of residues at node k whose cost is one more than that of the residues in R_k. The traditional parsimony algorithm can readily be extended so the Qs are computed at the same time as the Rs.

We have formulated parsimony in the context of rooted trees. However, the minimum cost for a tree in traditional parsimony is independent of where the root is located. In fact, the two edges below the root cannot both have substitutions in them in an optimal tree, for otherwise the assignment at the root could be changed to the assignment of one of the nodes below, with a reduction in cost. This means that, in principle, the root can be removed and costs counted along the edges of the unrooted tree. As it happens, it is easiest to count costs in a rooted tree, because the root defines a direction and hence a unique parent–daughter relationship for applying the parsimony algorithm. But the independence of root position means that the number of trees that have to be searched over is reduced.

Exercises

7.11 Show that the tradition parsimony algorithm gives the same cost as that for weighted parsimony using weights $S(a,a) = 0$ for all a, and $S(a,b) = 1$ for all $a \neq b$. (Hint: Show that, to obtain the minimal cost residues a of $S_k(a)$ at each node k, it suffices to keep the list R_k at each node.)

7.12 Show that the minimal cost with weighted parsimony is also independent of the position of the root, provided the substitution cost is a metric, i.e. satisfies $S(a,a) = 0$, symmetry $S(a,b) = S(b,a)$, and the triangle inequality $S(a,c) \leq S(a,b) + S(b,c)$, for all a,b,c.[3] (Hint: If there is a residue A at the root, and different residues B and C at the two daughters, show that the triangle inequality implies the cost cannot be minimal. Use the other two properties of a metric to show that the root can be moved to either of the daughter nodes without increase of cost.)

Selecting labelled branching patterns by branch and bound

We have seen that the search of trees with parsimony can be reduced because only unrooted trees need be considered. Nonetheless, the number of topologies swiftly becomes large as the number of leaves increases. For this reason some more efficient search strategy is needed than simple enumeration.

There are tree-searching methods that proceed stochastically; for instance, we can swop randomly chosen branches on a tree and choose the altered tree if it

[3] Sankoff & Cedergren [1983] assume their cost is a metric, but this is the only place where this property is needed.

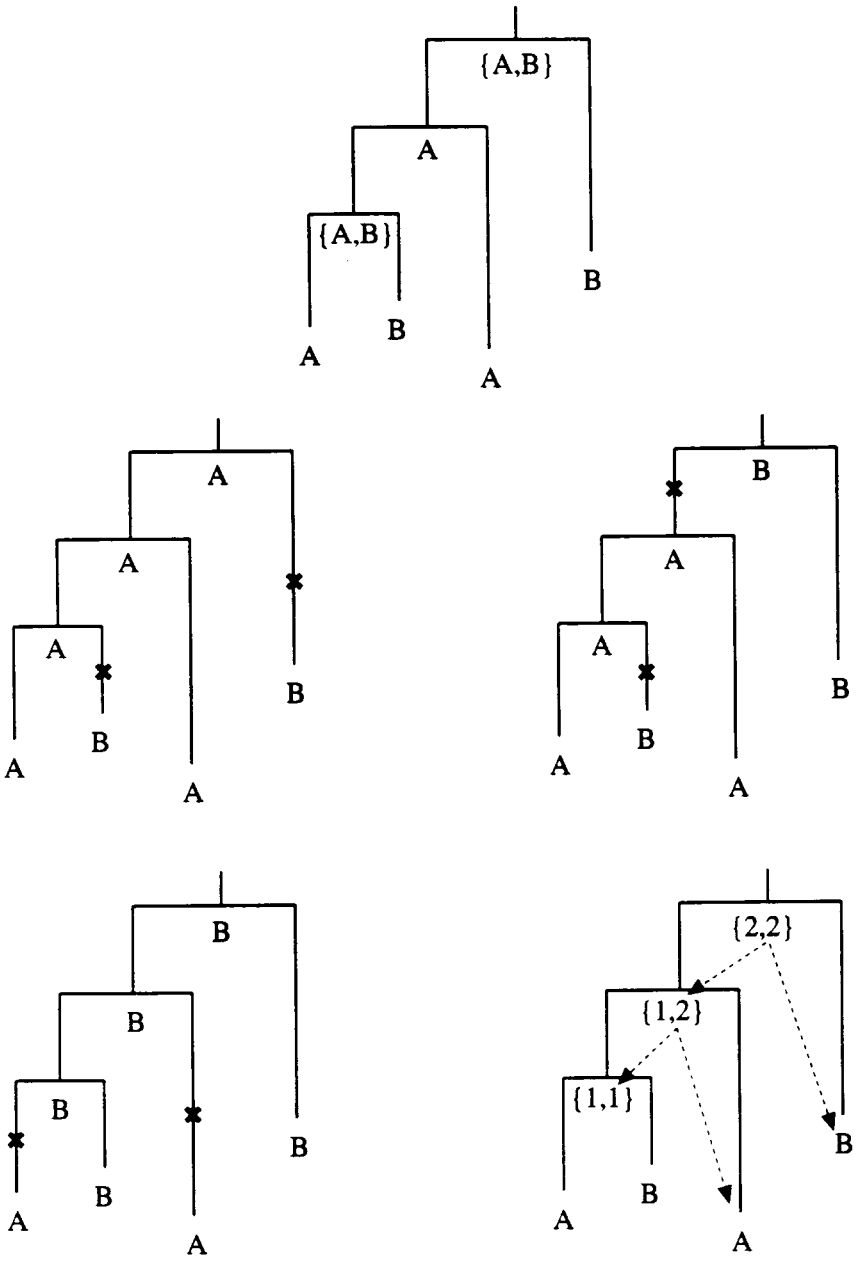

Figure 7.10 *Traditional parsimony with a cost of one for a substitution (marked by an 'X' on the edge), and zero cost otherwise. The sets R_k are shown in the top tree, the two middle trees show assignments of ancestral residues obtained by traceback using the sets R_k. The bottom left tree shows a further eligible set of ancestral residues that cannot be obtained this way, and the bottom right tree shows how this assignment would be obtained using the traceback for weighted parsimony (7.5).*

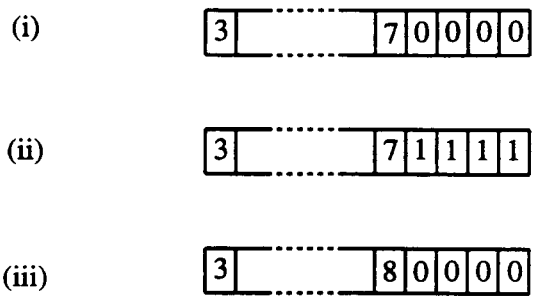

Figure 7.11 *A milometer (or odometer) used for counting unrooted trees.*

scores better than the current one. This is not guaranteed to find the overall best tree. Another strategy is to build up the tree by adding edges one at a time. Three sequences are chosen randomly and placed on an unrooted tree. Another sequence is then chosen and added to the edge that gives the best score for the tree of four sequences. A further randomly chosen sequence is added in the best-scoring position, and so on, until the tree is complete. This too is not guaranteed to find the overall best tree, and indeed adding the sequences in different orders can yield different final trees [Felsenstein 1981a].

With parsimony, there is an alternative strategy which is guaranteed to find the best tree; it exploits the fact that the number of substitutions in a tree can only be increased by adding an extra edge. The idea behind *branch and bound* is to begin systematically building trees with increasing numbers of leaves, but to abandon a particular avenue of tree building whenever the current incomplete tree has a cost exceeding the smallest cost obtained so far for a complete tree.

Let us enumerate all the unrooted trees by an array $[i_3][i_5][i_7]\ldots[i_{2n-5}]$, with each i_k taking values $1\ldots k$. The correspondence with trees is obtained as follows: Take the unrooted tree with the three sequences x^1, x^2 and x^3 and add an edge for x^4 on the edge labelled by i_3. Since this new edge divides a pre-existing edge in two, the total number of edges is now $3+2=5$. The value of i_5 determines which of these we add x^5 to, giving $5+2=7$ edges. And so on, up to x^n, which has $(2n-5)$ choices of position.

Now think of $[i_3][i_5][i_7]\ldots[i_{2n-5}]$ as a milometer (or odometer in the USA) on a car's dashboard. The rightmost numbers advance till they reach $2n-5$, when they go back to 1 and the next-to-rightmost array index clicks forward by 1. When the next-to-rightmost array index reaches $2n-7$ it starts again at 1 and the second-to-right array index clicks forward by 1. And so on.

This enumerates all trees with n leaves in a specified sequence, but we also want to count trees with fewer than n leaves, since we are going to build trees of

varying sizes. We therefore add a '0' to each counter, meaning that there is no edge of the order specified by the counter, and we let each index cycle from 0 to i_k. However, this will produce some meaningless values, because we cannot add an edge to a non-existent edge, i.e. we cannot have a non-zero counter to the right of a 0. Therefore, when we reach a situation with a row of 0s on the right, we have to advance them all simultaneously to '1' to make the next step (e.g. going from (i) to (ii) in Figure 7.11).

The process starts from the milometer setting [1][0][0]...[0]. Let the smallest cost so far for a complete tree be C. Whenever the cost of our current tree T is more than C, we know that T is not the optimal tree. But (here's the trick) if this happens when all the counters to the right of a given non-zero counter are 0, instead of advancing them all to '1' we can click the rightmost non-zero counter one forward. The reason for this is that the rightmost non-zero counter defines a tree with $k < n$ leaves, and adding more leaves can only increase the cost. We can therefore proceed to the next tree with k leaves (e.g. go from (i) to (iii) in Figure 7.11). This method can save a great deal of searching.

7.5 Assessing the trees: the bootstrap

The tree building algorithms we have described present us with a tree, or perhaps, in the case of parsimony, several optimal trees, but with no measure of how much they should be trusted. Felsenstein [1985] suggested using the bootstrap [Efron & Tibshirani 1993] as a method of assessing the significance of some phylogenetic feature, such as the segregation of a particular set of species on their own branch (a 'clade').

The bootstrap works as follows: Given a dataset consisting of an alignment of sequences, an artificial dataset of the same size is generated by picking columns from the alignment at random with replacement. (A given column in the original dataset can therefore appear several times in the artificial dataset.) The tree building algorithm is then applied to this new dataset, and the whole selection and tree building procedure is repeated some number of times, typically of the order of 1000 times. The frequency with which a chosen phylogenetic feature appears is taken to be a measure of the confidence we can have in this feature.

For certain probabilistic models, the bootstrap frequency of a phylogenetic feature F can be shown to approximate the posterior distribution $P(F|\text{data})$ (see p. 212). When the bootstrap is applied to a non-probabilistically formulated model, such as parsimony, it can be interpreted in terms of statistical hypothesis testing, though a rather more elaborate procedure than that given above may be needed to make the bootstrap conform to standard notions of confidence intervals [Efron, Halloran & Holmes 1996].

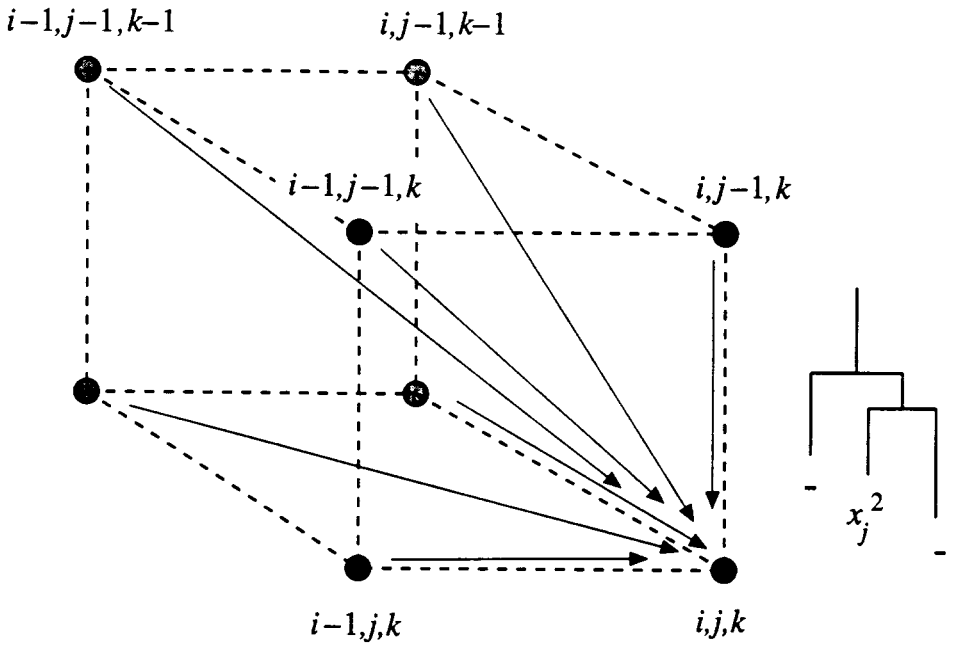

Figure 7.12 *Dynamic programming matrix for Sankoff & Cedergren's algorithm for 3 sequences. Each transition in the matrix is shown by an arrow. For 3 sequences there are 7 transitions at each point. Each has a cost assigned to it, which is the minimal cost, derived by the parsimony algorithm, of a tree whose leaves are defined by the transition, as follows: If 1 is subtracted from a coordinate, the relevant leaf is assigned the preceding character in the input sequence; if the coordinate is unchanged, its leaf is assigned a '-'. For instance, the transition from $(i, j-1, k)$ to (i, j, k) is assigned the tree shown in the figure.*

7.6 Simultaneous alignment and phylogeny

We turn now to the problem of simultaneously aligning sequences and finding a plausible phylogeny for them. There are two parsimony-type algorithms that tackle this problem, the first using a character-subsitution model of gaps, the second using affine gap penalties. Both find an optimal alignment given a tree; it is necessary to search over trees to find the overall optimum.

Sankoff & Cedergren's gap-substitution algorithm

Sankoff & Cedergren's algorithm is guaranteed to find ancestral sequences, and alignments of them and the leaf sequences, that together minimise a tree-based, parsimony-type cost [Sankoff & Cedergren 1983]. The algorithm is, in fact, a combination of two methods already introduced in this book (Figure 7.12). In Chapter 6, p. 141, a dynamic programming method was described for aligning a set of N sequences $x^1, x^2, \ldots, x^N$ [Sankoff & Cedergren 1983; Waterman 1995].

Replacing the max by a min (we use costs here rather than scores), the minimum cost $\alpha_{i_1,i_2,\ldots,i_N}$ of an alignment ending with $x^1_{i_1}, x^2_{i_2}, \ldots, x^N_{i_N}$ is

$$\alpha_{i_1,i_2,\ldots,i_N} = \min_{\Delta_1+\ldots+\Delta_N>0} \{\alpha_{i_1-\Delta_1,i_2-\Delta_2,\ldots,i_N-\Delta_N} + \sigma(\Delta_1 \cdot x^1_{i_1}, \Delta_2 \cdot x^2_{i_2}, \ldots, \Delta_N \cdot x^N_{i_N})\}. \tag{7.6}$$

where Δ_i is 0 or 1, and $\Delta_i \cdot x = x$ if $\Delta_i = 1$ and $\Delta_i \cdot x =$'-' if $\Delta_i = 0$. σ is the weighted parsimony cost for aligning a set of symbols of the extended alphabet. This cost can be calculated by an upward pass through the tree, using the weighted parsimony algorithm (p. 174), where $S(a,b)$ is now defined not when a,b are pairs of residues, but also when one or both is the gap symbol '-'.

Sankoff & Cedergren's procedure is therefore the following: When we reach $(i_1, i_2, \ldots, i_N)$ in the induction, each of the terms $\alpha_{i_1-\Delta_1,i_2-\Delta_2,\ldots,i_N-\Delta_N}$ in (7.6), for all $2^N - 1$ combinations of $\Delta_1, \ldots, \Delta_N$, will previously have been computed, and the calculation of $\sigma(\Delta_1 \cdot x^1_{i_1}, \Delta_2 \cdot x^2_{i_2}, \ldots, \Delta_N \cdot x^N_{i_N})$ can be achieved by an upward pass of the tree, requiring of the order of N steps (one step for each edge). The entire computation therefore requires of the order of $N(2n)^N$ steps, where n is the length of the sequences. Unfortunately, this is too large for more than half a dozen or so sequences of normal length (of the order of 100 residues).

Hein's affine cost algorithm

Hein's algorithm [Hein 1989a] uses an affine gap cost which is more realistic than the simple substitution treatment of gaps. It is also much faster than Sankoff & Cedergren's algorithm in most realistic situations, fast enough in fact to allow a search over tree topologies for modest-sized sets of sequences. It is the only current practical algorithm able to align sequences and explore alternative phylogenies effectively. The price paid for these very considerable gains is that the algorithm makes a simplifying assumption in the choice of ancestral sequences which does not always lead to the overall most parsimonious choices.

Suppose that we are given a tree. Recall that the algorithm for traditional parsimony ascends the tree, assigning a list of possible residues to each node. These residues are just those that minimise the number of substitutions along the edges to the two daughter nodes. In this case it is possible to find the minimal number of substitutions for the whole tree by minimising at each node. This same procedure is used in Hein's algorithm: in the upward pass through the tree, the only sequences that are considered at a node are those that have the minimum cost, given the sequences at the two daughter nodes. We shall see later that, unlike traditional parsimony, this procedure is not guaranteed to find the minimum cost for the whole tree. But first, let us see how the minimum cost sequences at each node are determined.

The aim is to find sequences z at a given node aligned to both of the sequences

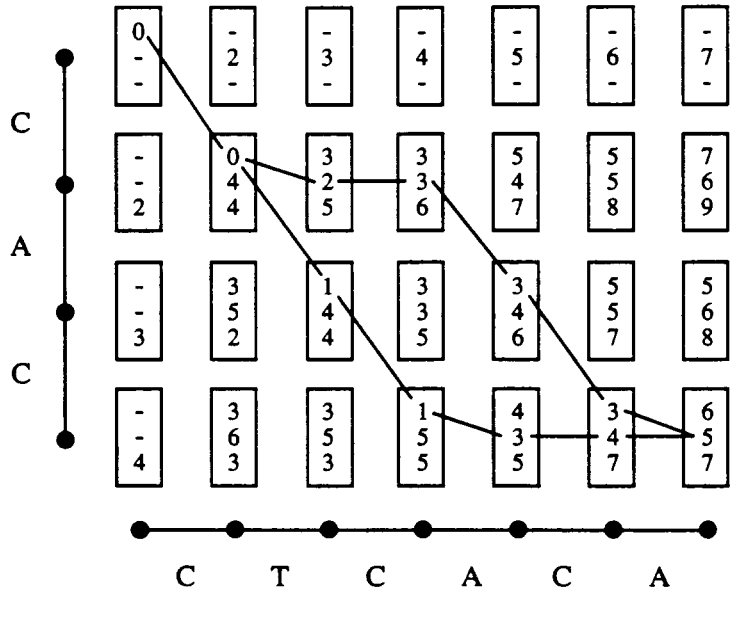

Figure 7.13 *The dynamic programming matrix for two sequences, showing cells with V^M, V^X and V^Y written in this order, from top to bottom. Optimal paths are shown as lines between the cells. Here $d = 2$, $e = 1$, and there is a mismatch cost of 1. Note that we include costs arising from a gap in one sequence followed by a gap in the other (eg the entry $V^Y = 4$ in the second cell from the left in the second row down, which arises from $V^X = 2$ in the cell above), even though such matches are non-optimal.*

x and y at the daughter nodes and satisfying

$$S(x,z) + S(z,y) = S(x,y), \tag{7.7}$$

where S here denotes the total cost for a given alignment of two sequences. Assuming a mismatch cost of one, with zero cost otherwise, (7.7) can be satisfied at any site if z shares a residue at each site with either x or y (or with both x and y when they have the same residue). Hein's algorithm can also be extended to general weighted parsimony (see Exercise 7.13).

We have not yet shown that sequences z satisfying (7.7) can be found, because we have to deal with gaps. To do this, we use the dynamic programming method for affine gaps described in Chapter 2. Let $V^{\mathrm{M}}(i,j)$, $V^{\mathrm{X}}(i,j)$, $V^{\mathrm{Y}}(i,j)$ denote the minimum costs for alignments up to position i in sequence x, j in sequence y, in the cases where (1) the ith residue in x is aligned to the jth in y, (2) the ith residue in x is aligned to a gap in y, and (3) the jth residue in y is aligned to a gap in x. These correspond to Viterbi costs up to the match state $\mathrm{M}(i,j)$, and insert states X and Y, respectively. We write the three numbers V^{M}, V^{X} and V^{Y} in the (i,j)th cell in the dynamic programming matrix (Figure 7.13). Let the affine gap

cost for a gap of length k be $d+(k-1)e$, where $e \leq d$. Then the recursion is

$$\begin{aligned} V^{\mathrm{M}}(i,j) &= \min\{V^{\mathrm{M}}(i-1,j-1), V^{\mathrm{X}}(i-1,j-1), V^{\mathrm{Y}}(i-1,j-1)\} \\ &\quad + S(x_i, y_j), \\ V^{\mathrm{X}}(i,j) &= \min\{V^{\mathrm{M}}(i-1,j)+d, V^{\mathrm{X}}(i-1,j)+e\}, \\ V^{\mathrm{Y}}(i,j) &= \min\{V^{\mathrm{M}}(i,j-1)+d, V^{\mathrm{Y}}(i,j-1)+e\}. \end{aligned} \tag{7.8}$$

Here we assume that the mismatch cost is less than $2e$, which ensures that an optimal alignment will never have a gap in one sequence immediately followed by a gap in the other, e.g. $\begin{smallmatrix}\mathtt{TTAC--}\\ \mathtt{TT--GG}\end{smallmatrix}$, but will prefer to match residues, e.g. $\begin{smallmatrix}\mathtt{TTAC}\\ \mathtt{TTGG}\end{smallmatrix}$ in this example.

Let us mark all the transitions that occur on paths that give the minimal cost (e.g. those marked in Figure 7.13). Any path that we piece together using these transitions will give an optimal alignment of x and y. Suppose now that x and y are the sequences at the two daughter nodes of a node n. Any path using our marked transitions also serves to define eligible ancestral sequences at n, as follows. If a transition corresponds to a match of two residues in x and y, we choose one of these residues for the ancestral sequence. If a transition corresponds to a match between a gap and a residue, we choose either a gap or a residue in the ancestral sequence.

This will yield a sequence z aligned to x and y. From the way z was constructed, it is clear that if, at some site, both x and y have a residue, then z shares a residue with either x or y (or possibly both of them). Thus equal contributions will be made to the two sides of (7.7). The same will be true when either x or y has a gap provided we take some care with gap-opening costs. In fact, our recipe for making ancestral sequences needs the following extra rule: If a block of consecutive gaps in one sequence occurs on a path, and these are aligned to a set of residues in the other sequence, then the ancestral sequence must either skip this entire set of residues or include them all.

For instance, given the two sequences CAC and CTCACA (see Figure 7.13), the sequence CTC can be derived by following the lower path in the matrix, corresponding to the alignment $\begin{smallmatrix}\mathtt{CAC---}\\ \mathtt{CTCACA}\end{smallmatrix}$, choosing a T in the second position, and skipping the block of three gaps. It is a possible ancestral sequence because it can be aligned to CAC by $\begin{smallmatrix}\mathtt{CTC}\\ \mathtt{CAC}\end{smallmatrix}$ with a cost of one for the A,T mismatch, and to CTCACA by $\begin{smallmatrix}\mathtt{CTC---}\\ \mathtt{CTCACA}\end{smallmatrix}$ with a cost of $d+2e$. The sum of these two costs, $d+2e+1$, is the cost of the original alignment $\begin{smallmatrix}\mathtt{CAC---}\\ \mathtt{CTCACA}\end{smallmatrix}$. Similarly CACACA is another eligible ancestral sequence, derived by choosing the residues from the block that are matched to the gaps. What is not allowed is to use only some of these residues. For instance, CACAC is not an ancestral sequence. In fact, optimal alignments to the daughter sequences are $\begin{smallmatrix}\mathtt{CACAC}\\ \mathtt{CAC--}\end{smallmatrix}$, with cost $d+e$, and $\begin{smallmatrix}\mathtt{CACAC-}\\ \mathtt{CTCACA}\end{smallmatrix}$, with cpst $d+1$, and both include gap-opening terms. The sum of both costs, $2d+e+1$, exceeds $d+2e+1$, the cost of the original alignment, since we are assuming $d > e$.

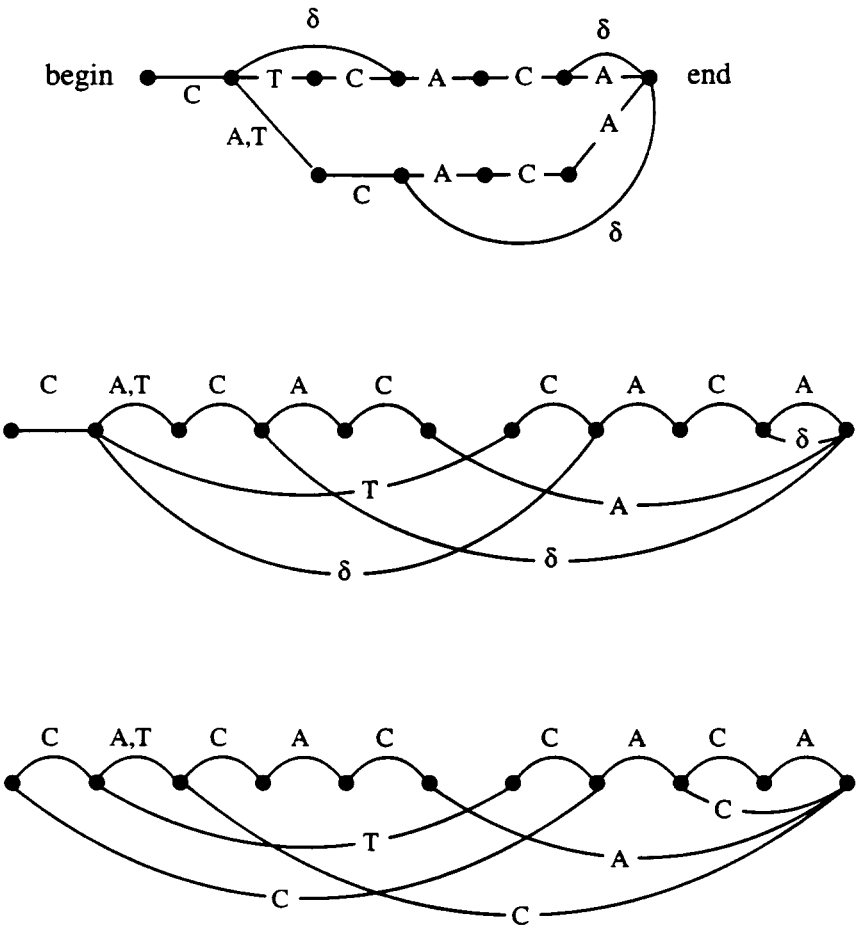

Figure 7.14 *The sequence graph derived from the paths through the dynamic programming matrix in Figure 7.13. Top: the graph, with its dummy edges (marked by a δ). Middle: the same graph, with its nodes arranged in a line. Bottom: the dummy edges have been replaced by an edge that goes back to the preceding vertex and emits the residues attached to that edge.*

We now formalise the idea of following paths through the dynamic programming matrix by deriving a graph from this matrix. Whenever any of the three entries of a cell of the dynamic programming matrix is used by an optimal path, we represent it by a vertex in the graph. It is important to note that different entries in the same cell need different vertices. The situation in Figure 7.13 shows why: The two optimal paths cross in the penultimate cell, one using the M state with cost 3, and the other using the X state with cost 4. If we switched paths in this cell, we would lose track of whether we were opening or extending a gap (see Altschul & Erickson [1986]).

The directed edges of the graph are the transitions that occur in an optimal alignment. We assign residues to all these edges: if a transition in the matrix ends in the match of two residues, then both residues are attached to the edge; if there is a match of a residue and a gap, only that one residue is attached to

the edge. Finally, we add 'dummy edges' corresponding to consecutive blocks of two or more gaps. These run from the vertex at the start of the block to that at the end of block and are assigned no residues.

We now consider paths through this graph, running from the initial point to the point matching the last residues in each sequence. The rule is that any path emits the symbols on the edges it uses, choosing one of the symbols if more than one is available. It's easy to see that any path through the graph emits an eligible sequence. The graph will be referred to as a *sequence graph.*

This construction applies to the case where each of the two daughters is a leaf of the tree and so has a single sequence associated to it. What happens as we ascend the tree, and the daughters can have many eligible sequences assigned to them? Hein's ingenious idea is to carry out exactly the same construction, but with graphs rather than sequences as the objects to be matched in the dynamic programming matrix.

To achieve this, we first stretch out each graph so its vertices lie in a line (middle diagram in Figure 7.14); this can always be done so that all the edges point in the same direction. Suppose we have two graphs, G_1 and G_2. Again, we keep track of the values of V^{M}, V^{X} and V^{Y} in each cell. However, instead of considering transitions from the preceding residue in the sequences, we now define 'preceding' by the incoming edges in the graph. When these include a dummy edge, we can skip back to the vertex at its start, and the preceding non-dummy edge then defines a preceding vertex. Note that, because dummy edges span the whole of a block of gaps, and because the condition that a mismatch cost is less than $2e$ excludes a block in one sequence following a block in the other, there cannot be a chain of consecutive dummy edges. Thus a combinatorial explosion in the number of preceding nodes cannot occur.

The procedure for dummy edges can be carried out by first modifying each sequence graph, removing all the dummy edges (marked δ in Figure 7.14) and replacing each of them by an edge going one step back beyond the start of the dummy edge and carrying the symbols associated to that preceding edge. The middle and bottom graphs of Figure 7.14 show how this replacement works. Now the transitions in the dynamic programming matrix are easily described: they are just those obtained by following edges in the modified G_1 or G_2 (see Figure 7.15).

Having defined the values of V in each cell in the matrix, the optimal paths are defined as before by backtracking, and the next sequence graph G_3 is constructed. There is one new feature. The edges in a sequence graph can have several symbols associated to them, these being the sets R_k of the traditional parsimony algorithm. The same procedure as for traditional parsimony governs the combining of sets of symbols: If V^{M} is defined from an edge in G_1 and an edge in G_2, and if there is a shared symbol (i.e. if there was no mismatch cost in the path using both edges), then only the shared symbols are attached to the derived edge

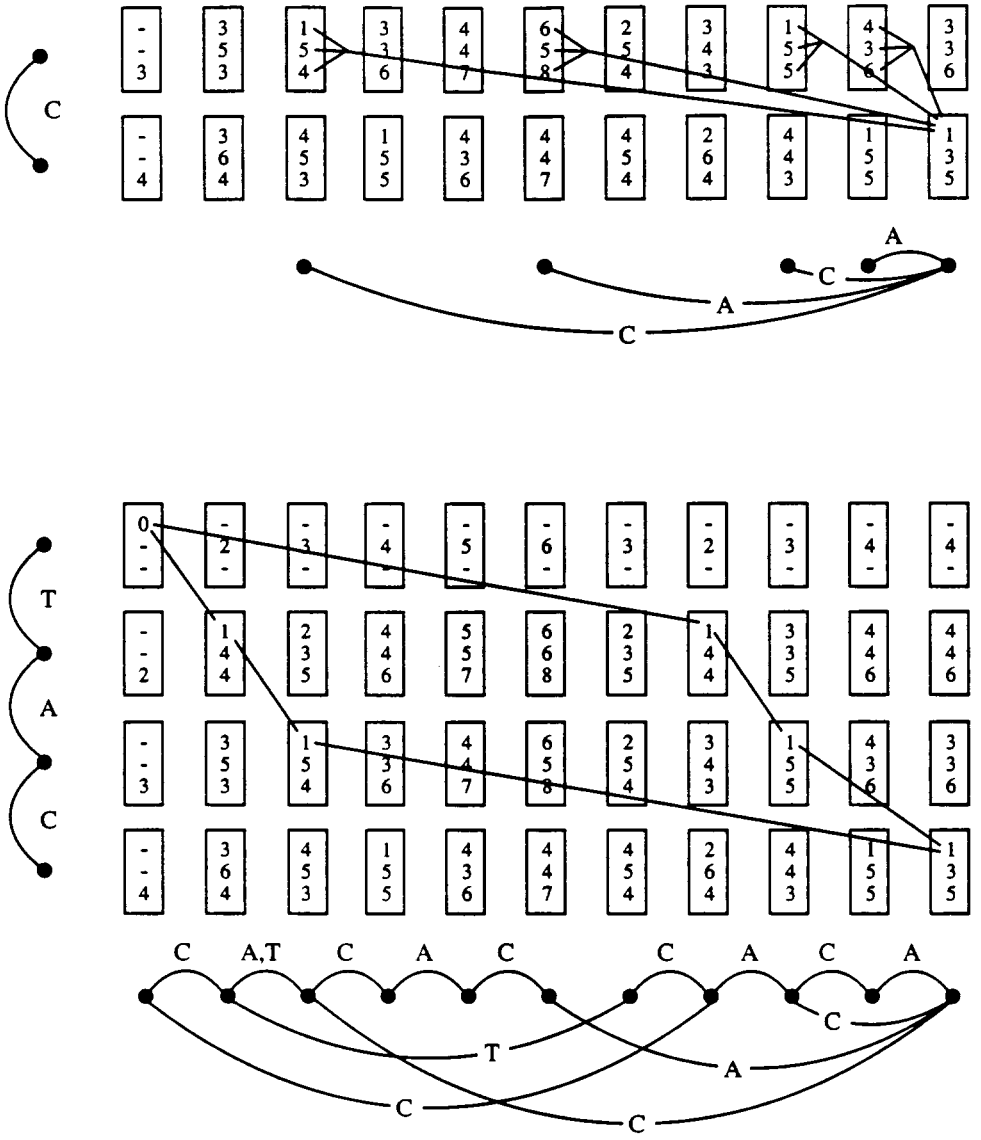

Figure 7.15 *The dynamic programming matrix for a sequence graph, the bottom graph in Figure 7.14, against the sequence* TAC. *The sequence graph in this matrix generates possible ancestral sequences for the top node of the tree in Figure 7.16. The values of V^M, V^X and V^Y in a cell are determined by taking the minimum over all 'preceding' vertices, these being the vertices that can be reached by an edge going back from the current vertex. This is illustrated in the figure above for the computation of a value of V^M in a cell.*

in G_3. If there is no shared symbol, the derived edge acquires all the symbols from both edges.

We can now carry out the recursion (7.8) on the matrix. The optimal paths through the matrix define another graph, and so we continue, ascending the tree until the root is reached. We can then descend the tree, reconstructing the daughter sequences corresponding to a given ancestral sequence. To do this, we follow

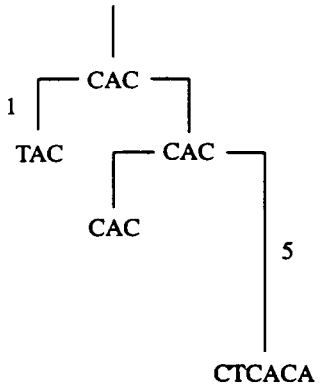

Figure 7.16 *Possible ancestral sequences for the leaf sequences* TAC, CAC, CTCACA, *given the tree shown in the figure.*

the sequence of edges in each daughter graph as the ancestral sequence is traced, choosing symbols in the daughters that are compatible with those in the ancestral sequence. If a delete, or a succession of deletes, skips successive nodes in one of the daughter graphs, the skipped edges must be filled in, with arbitrary choices of symbols. For instance, in tracing the ancestral sequence CAC through the lower path in Figure 7.13, the first three symbols CTC... of the daughter sequence CTCACA are generated, and the last three symbols ...ACA must be added, even though they are skipped in the ancestral path by using a dummy edge (the δ edge of length 3 shown at the top of Figure 7.14). Possible ancestral sequences for a tree are shown in Figure 7.16; the sequence graphs for this tree are those shown in Figure 7.13 and Figure 7.15.

We have now described how sequences are aligned on a given tree. It is also necessary to search through trees, for which Hein [1989b] proposes his own efficient search algorithm. The entire procedure will be manageable if the sequence graphs do not become too complicated. If we have to include most of the transitions in the dynamical programming matrix, the computation will increase in complexity like Sankoff & Cedergren's. The assumption is that most alignments will have a few main routes giving the minimal cost. This will be true if the sequences are similar enough, for then there will be long stretches of unambiguous matches which define a single path through the matrix.

Exercise

7.13 Hein's algorithm can be extended to general weights $S(a,b)$ by attaching a set of minimal costs $S_k(a)$ (as in the weighted parsimony algorithm) to each edge in a sequence graph instead of the set R_k. Show that (7.7) can be satisfied by having z share a residue with x or y provided that $S(a,a) = 0$ for all a. Evaluate the minimal costs (assuming a nucleic acid alphabet) for the sequence graphs shown in Figure 7.13, Figure 7.14 and Figure 7.15.

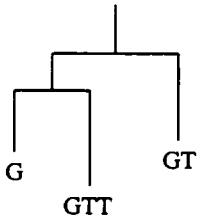

Figure 7.17 *A case where Hein's rule of for choosing optimal ancestral sequences fails to produce the optimal overall assignment to ancestors.*

Limitations of Hein's model

Now let us return to Hein's procedure of taking the minimal cost sequences at each node in the upward pass. To see how this can fail to give an overall optimum for the tree, suppose the cost for a gap of length k is $13+3(k-1)$ and the mismatch cost is 4 (values used by Hein in an example alignment of 5S RNAs). The eligible ancestral sequences for G and GTT are just G and GTT themselves; each requires a consecutive pair of gaps to one of its daughters, with cost $13+3=16$. The sequence GT is not an eligible ancestral sequence, since it requires single gaps in both alignments, with a total cost of $2 \times 13 = 26$, i.e. $\frac{\mathtt{G-}}{\mathtt{GT}}$ and $\frac{\mathtt{GT-}}{\mathtt{GTT}}$. But suppose we have a tree in which the root branches to the ancestor of G and GTT, and also to a third leaf with sequence GT (see Figure 7.17). Then the total tree cost for the ineligible ancestor GT is smaller because two gaps of size one are required in the tree in that case, as opposed to gaps of sizes one and two when either eligible ancestor is used.

Warning B. Schwikowski has recently showed that the dummy edge construction is flawed. An alternative procedure that uses Hein's sequence graphs but avoids this pitfall is described in Schwikowski, B. and Vingron, M. 1997. The deferred path heuristic for the generalized tree alignment problem. *Journal of Computational Biology* 4:415-431.

7.7 Further reading

Parsimony was first formulated by Edwards & Cavalli-Sforza [1963; 1964] in the case of continuous parameters, where it amounts to finding a minimum-length tree joining points in Euclidean space. Counting algorithms for parsimony based on sequence data or other discrete variables were introduced by Camin & Sokal [1965], Eck & Dayhoff [1966], Fitch [1971] and others. The combination of the simplicity of these algorithms and the richness of sequence data has combined to make these methods very popular.

Parsimony is sometimes alleged to be a direct philosophical descendant of Occam's razor, and to be free of specific evolutionary assumptions, e.g. 'The

use of parsimony in phylogenetic systematics is no different from its use in any other branch of biology or any other science, [and] does not invoke any particular evolutionary mechanism' [Brooks & McLennan 1991, p. 65]. For an instructive debunking of this notion, see Edwards [1996]. See also Section 8.6 for an interpretation of parsimony that relates it to probabilistic models described in this book.

Phylogenetic distance methods were also first described by Edwards & Cavalli-Sforza in their papers mentioned above. They proposed a least-squares match between the observed distances and the summed lengths of a tree. Neighbour-joining can also be given a least-squares interpretation: in this case, the observed distances are compared with those in a simplified tree [Saitou & Nei 1987]. There are many other distance methods besides those discussed here; to mention only one other, that of Fitch & Margoliash [1967a] combines clustering with the distance definition of (7.3).

There are a number of mathematical results on trees with additive lengths. In addition to Studier & Keppler's theorem given in Section 7.8, Buneman [1971] has shown that, when a set of distances satisfies the four-point condition (p. 172), a tree and a set of edge lengths can be found that generate these distances as the sum of edge lengths.

Horizontal transfer of genetic material gives an interesting twist to phylogeny. At first sight, it prevents the use of simple tree structures, since a recombinational event would seem to create a link between a sequence and its two ancestors and thereby give rise a loop. However, the fragments on either side of a recombination point each have a single parent, and the genome may be described as a concatenation of segments, each with its own tree [Hein 1993]. Recombinational events are likely to be particularly important in viruses and prokaryotes, where horizontal transfer is frequent. (The recombination that occurs in diploid 'family trees' is of course even more frequent, but it requires a different kind of model, being dominated by crossing-over events with little evolution of sequence.)

A generalisation of trees has been proposed by Bandelt & Dress [1992]. They showed how to build networks from distance data that branch like a tree where the evidence for a topology is strong and that generate a mesh covering regions of ambiguity.

Useful general references for phylogeny are Waterman [1995], Swofford & Olsen [1996]; for recent reviews see Saitou [1996] and Felsenstein [1996].

7.8 Appendix: proof of neighbour-joining theorem

For completeness, and because it is a pleasing mathematical result, we include the proof by Studier & Keppler [1988] that leaves with minimal neighbour-joining distance are neighbours. This ensures that a tree with additive lengths will be

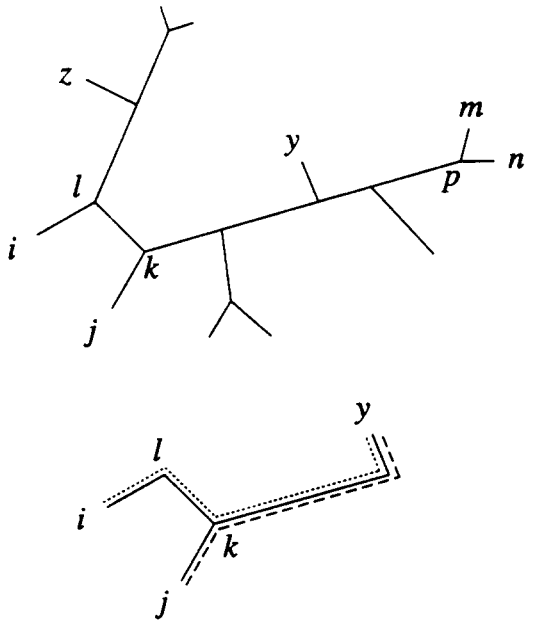

Figure 7.18 *Above: If the leaves i, j are not neighbours, there are at least two nodes on the path joining them, here shown as k and l. The branch from k that does not go to i or j is called L_k and is shown here with a pair of neighbours, m and n, on it. The branch from l is called L_l, and has a branch to a leaf z. Below: The path from i to y (dots) and from j to y (dashes) are shown, giving a simple visual proof that $d_{iy} + d_{jy} = d_{ij} + 2d_{ky}$.*

correctly reconstructed by neighbour-joining. A recent paper extends this result to show that neighbour-joining also correctly reconstructs trees where additivity only holds approximately [Atteson 1997].

Theorem: For a tree with additive lengths, D_{ij} minimal implies i,j are neighbouring leaves.

Proof: Suppose the smallest D is D_{ij}, and suppose furthermore that i and j are not neighbouring leaves. We seek a contradiction.

Since i and j are not neighbours, there must be at least two nodes on the path connecting them (see Figure 7.18). Call these nodes k, l, and let L_k be the set of leaves which derive from the third branch from k, i.e. not the edge towards i or j, and let L_l be the equivalent set for l. Let m and n be a pair of neighbouring leaves in L_k with joining node p (if no such pair exists, an alternative argument is available; see Exercise 7.14). Let d_{uv} denote the summed edge lengths of the path connecting any two nodes u, v. By additivity, this is the correct distance d_{uv} when they are both leaves. For any y in L_k, it is clear that $d_{iy} + d_{jy} = d_{ij} + 2d_{ky}$ (see lower figure in Figure 7.18). Similarly $d_{my} + d_{ny} = d_{mn} + 2d_{py}$. Thus

$$d_{iy} + d_{jy} - d_{my} - d_{ny} = d_{ij} + 2d_{ky} - 2d_{py} - d_{mn}. \tag{7.9}$$

Likewise, for z in L_l, we find

$$d_{iz} + d_{jz} - d_{mz} - d_{nz} = d_{ij} - d_{mn} - 2d_{pk} - d_{lk}. \tag{7.10}$$

From the definition of D_{ij},

$$D_{ij} - D_{mn} = d_{ij} - d_{mn} - \frac{1}{N-2}\left(\sum_{\text{all leaves } u} d_{iu} + d_{ju} - d_{mu} - d_{nu}\right),$$

and it is easy to check from (7.9) and (7.10) that the coefficients of d_{ij} and d_{mn}, summed over all leaves u in the tree, are both $(N-2)$ (see Exercise 7.15). Thus the term $d_{ij} - d_{mn}$ cancels, and we can write

$$D_{ij} - D_{mn} = \frac{1}{N-2}\left(\sum_{y \text{ in } L_k} (2d_{py} - 2d_{ky}) + \sum_{z \text{ in } L_l} (2d_{pk} + d_{lk})\right) + C$$

where C is the sum of all the extra positive terms coming from other branches on the path between i and j besides k and l. Letting $|L_l|$ and $|L_k|$ denote the numbers of nodes in L_l, L_k, respectively, and the using the fact that $d_{py} - d_{ky} > -d_{pk}$,

$$D_{ij} - D_{mn} > 2d_{pk}(|L_l| - |L_k|)/(N-2).$$

We must have $D_{mn} > D_{ij}$, since D_{ij} is the minimum, so $|L_l| < |L_k|$. But the argument can be applied with the two nodes l and k reversed, so we must also have $|L_k| < |L_l|$. Hence the assumption is false, and i, j are neighbouring leaves.

Exercises

7.14 If the branch from k has only a single leaf m (so it is not possible to find a pair of neighbours in L_k), show that the presence of other nodes besides k on the path from i to j implies $D_{ij} > D_{jm}$, contradicting the assumption that D_{ij} is the minimum.

7.15 Show that the term $2d_{py}$ is absent in (7.9) when $y = m$ or $y = n$. Show that this means that the term $2d_{py}$ can be included in the sum $\sum(2d_{py} - 2d_{ky})$ in (7.11) for all y in L_k, including $y = m$ and $y = n$, provided we subtract $2d_{pm} + 2d_{pn}$ from the sum. Show that the term d_{mn} then cancels, and also check the case where $y = i$ and $y = j$.

8

Probabilistic approaches to phylogeny

8.1 Introduction

Our goal in this chapter is to formulate probabilistic models for phylogeny and show how trees can be inferred from sets of sequences, either by maximum likelihood or by sampling methods. We also review the phylogenetic methods of the previous chapter, and show that they often have probabilistic interpretations, though they are not usually presented this way.

Overview of the probabilistic approach to phylogeny

The basic aim of probability-based phylogeny is to rank trees either according to their likelihood $P(\text{data}|\text{tree})$, or, if we are taking a more Bayesian view, according to their posterior probability $P(\text{tree}|\text{data})$. There may be subsidiary aims, such as finding the likelihood or posterior probability of some particular taxonomic feature, such as a grouping of a set of organisms on a single branch. To achieve any of these aims, we must be able to define and compute $P(x^\bullet|T,t_\bullet)$, the probability of a set of data given a tree. Here the data are a set of n sequences x^j for $j = 1 \ldots n$, which we write compactly as $x^\bullet$. T is a tree with n leaves with sequence j at leaf j, and the $t_\bullet$ are the edge lengths of the tree. To define $P(x^\bullet|T,t_\bullet)$ we need a model of evolution, i.e. of the mutation and selection events that change sequences along the edges of a tree.

Let us assume that we can define a probability $P(x|y,t)$ for an ancestral sequence y to evolve to a sequence x along an edge of length t. The probability of T with a specific set of ancestors assigned to its nodes can then be calculated by multiplying all the evolutionary probabilities, one for each edge of the tree. For instance, for the tree shown in Figure 8.1 the probability would be

$$P(x^1,\ldots,x^5|T,t_\bullet) = P(x^1|x^4,t_1)P(x^2|x^4,t_2)P(x^3|x^5,t_3)P(x^4|x^5,t_4)P(x^5),$$

where $P(x^5)$ denotes the probability of x^5 occurring at the root of the tree. In general (apart from laboratory evolution experiments, like those of Hillis *et al.* [1992]) the ancestral sequences will be unknown, and to obtain the probability $P(x^1,\ldots,x^3|T,t_\bullet)$ of the known sequences for the given tree we need to sum over

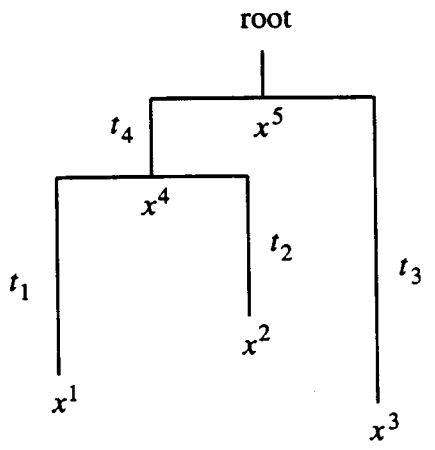

Figure 8.1 *An example of a tree with three sequences.*

all the possible ancestors x^4, x^5. This is similar to summing over all the different paths in a HMM to obtain the probability of the observed data (see Chapter 3).

Given this model, we can seek the maximum likelihood tree, namely the tree with topology T and edge lengths $t_\bullet$ that maximises $P(x^\bullet|T,t_\bullet)$. Finding this maximum requires: (1) a search over tree topologies, with the order of assignment of sequences at the leaves specified; (2) for each topology, a search over all possible lengths of edges $t_\bullet$.

As we have seen (p. 164), there are $(2n-3)!!$ rooted binary trees with n leaves, and this number grows very large for more than half a dozen sequences. An efficient search procedure (e.g. p. 176 and Felsenstein [1981a]) is therefore required to carry out (1). Part (2), maximising the likelihood of edge lengths, can be achieved by a variety of optimisation techniques (Section 8.3).

An alternative strategy is to search stochastically over trees by sampling from the posterior distribution $P(T,t_\bullet|x^\bullet)$ (Section 8.4). This has only been explored recently, but the method is very promising.

8.2 Probabilistic models of evolution

We have not yet specified the form of $P(x|y,t)$, the probability of a sequence x arising from an ancestral sequence y over an edge of length t. For this we need a model of evolution. We know that, in the course of evolution, residues are substituted by others, that deletion and insertion of groups of residues occur, and that there are more complex constraints imposed by structures of nucleic acids and proteins. Later we shall consider models for deletions and insertions, but to begin with we make some radical simplifying assumptions: that every site of the given data sequences can be treated as independent and that deletions and

insertions do not occur. Our sequences therefore form an ungapped alignment, with independent evolution at each site.

Let $P(b|a,t)$ denote the probability of a residue a having being substituted by a residue b over an edge length t. Then our assumption implies that for two aligned, gapless sequences x and y, $P(x|y,t) = \prod_u P(x_u|y_u,t)$, where u indexes sites in the alignment.

Let us look now at possible forms for the substitution probabilities $P(b|a,t)$, for residues a and b. Given a residue alphabet of size K, we can write these as a $K \times K$ matrix that depends on t, and which we denote by $S(t)$:

$$S(t) = \begin{pmatrix} P(A_1|A_1,t) & P(A_2|A_1,t) & \dots & P(A_K|A_1,t) \\ P(A_1|A_2,t) & P(A_2|A_2,t) & \dots & P(A_K|A_2,t) \\ \dots & \dots & \dots & \dots \\ P(A_1|A_K,t) & P(A_2|A_K,t) & \dots & P(A_K|A_K,t) \end{pmatrix}.$$

For several important families of substitution matrices, the family is *multiplicative*, in the sense that

$$S(t)S(s) = S(t+s) \tag{8.1}$$

for all values of the lengths s and t. This is equivalent to saying that the substitution probabilities satisfy

$$\sum_b P(a|b,t)P(b|c,s) = P(a|c,s+t)$$

for all a, c, s and t. If we adopt a viewpoint in which t is regarded as a 'time' variable,[1] then multiplicativity is a consequence of the substitution process being Markovian and stationary, the latter meaning that the probability of substituting a at time t by b at time s depends only on the time interval $(s-t)$ (Exercise 8.2).

For nucleotide sequences, one model is that of Jukes & Cantor [1969]. This assumes that the matrix, R, of *rates* of substitution takes the form

$$\begin{array}{c} \\ \mathtt{A} \\ \mathtt{C} \\ \mathtt{G} \\ \mathtt{T} \end{array} \begin{array}{c} \begin{array}{cccc} \mathtt{A} & \mathtt{C} & \mathtt{G} & \mathtt{T} \end{array} \\ \begin{pmatrix} -3\alpha & \alpha & \alpha & \alpha \\ \alpha & -3\alpha & \alpha & \alpha \\ \alpha & \alpha & -3\alpha & \alpha \\ \alpha & \alpha & \alpha & -3\alpha \end{pmatrix} \end{array}, \tag{8.2}$$

which means that all nucleotides undergo transitions at the same rate α. The substitution matrix for a short time $S(\varepsilon)$ is approximately given by $S(\varepsilon) \simeq (I + R\varepsilon)$, where I is the identity matrix with ones down the diagonal and zeros elsewhere.

[1] For instance, t might be proportional to mutation rate × evolutionary time.

Thus

$$I + R\varepsilon = \begin{pmatrix} 1-3\alpha\varepsilon & \alpha\varepsilon & \alpha\varepsilon & \alpha\varepsilon \\ \alpha\varepsilon & 1-3\alpha\varepsilon & \alpha\varepsilon & \alpha\varepsilon \\ \alpha\varepsilon & \alpha\varepsilon & 1-3\alpha\varepsilon & \alpha\varepsilon \\ \alpha\varepsilon & \alpha\varepsilon & \alpha\varepsilon & 1-3\alpha\varepsilon \end{pmatrix}.$$

By multiplicativity, $S(t+\varepsilon) = S(t)S(\varepsilon) \simeq S(t)(I + R\varepsilon)$. We can write this as $(S(t+\varepsilon) - S(t))/\varepsilon \simeq S(t)R$, and in the limit of small ε we get $S'(t) = S(t)R$. From this we can derive a substitution matrix for time t. The symmetry of the rate matrix suggests that we try giving $S(t)$ the following form:

$$S(t) = \begin{pmatrix} r_t & s_t & s_t & s_t \\ s_t & r_t & s_t & s_t \\ s_t & s_t & r_t & s_t \\ s_t & s_t & s_t & r_t \end{pmatrix}. \tag{8.3}$$

Substituting this into $S'(t) = S(t)R$, we get the equations

$$\begin{aligned} \dot{r} &= -3\alpha r + 3\alpha s, \\ \dot{s} &= -\alpha s + \alpha r, \end{aligned}$$

and we easily check that these are satisfied by

$$\begin{aligned} r_t &= \tfrac{1}{4}\left(1 + 3e^{-4\alpha t}\right), \\ s_t &= \tfrac{1}{4}\left(1 - e^{-4\alpha t}\right). \end{aligned} \tag{8.4}$$

The matrix (8.3) with these values of r_t and s_t constitutes the *Jukes–Cantor model*. Note that, when $t = \infty$, $r_t = s_t = \frac{1}{4}$. This means that the nucleotide equilibrium frequencies implied by the model are $q_A = q_C = q_G = q_T = \frac{1}{4}$.

The Jukes–Cantor model does not capture some important features of nucleotide substitution. For instance, transitions, namely purine to purine or pyrimidine to pyrimidine substitutions, are more common than transversions, which change the type of nucleotide.[2] To account for this, Kimura [1980] proposed a model with the rate matrix

$$\begin{pmatrix} -2\beta-\alpha & \beta & \alpha & \beta \\ \beta & -2\beta-\alpha & \beta & \alpha \\ \alpha & \beta & -2\beta-\alpha & \beta \\ \beta & \alpha & \beta & -2\beta-\alpha \end{pmatrix}. \tag{8.5}$$

This can be integrated, by the same procedure we used with the Jukes–Cantor

[2] Thus the transitions are A ↔ G, C ↔ T, and the transversions are A ↔ T, G ↔ T, A ↔ C, and C ↔ G

model, to give the general time-dependent form

$$S(t) = \begin{pmatrix} r_t & s_t & u_t & s_t \\ s_t & r_t & s_t & u_t \\ u_t & s_t & r_t & s_t \\ s_t & u_t & s_t & r_t \end{pmatrix}, \tag{8.6}$$

where

$$\begin{aligned} s_t &= \tfrac{1}{4}\left(1-e^{-4\beta t}\right), \\ u_t &= \tfrac{1}{4}\left(1+e^{-4\beta t}-2e^{-2(\alpha+\beta)t}\right), \\ r_t &= 1-2s_t-u_t. \end{aligned}$$

This model, though widely used, is still far from realistic, as its equilibrium frequencies are equal, $q_A = q_C = q_G = q_T = \frac{1}{4}$, whereas many organisms show strong bias in their AT to GC ratio. For a model which allows for this as well as inequality of transitions and transversions, see Hasegawa, Kishino & Yano [1985].

Turning to protein sequences, we saw in Section 2.8 that the PAM matrices of conditional probabilities for integers n are defined by $S(n) = S(1)^n$, i.e. by raising the PAM1 matrix to the nth power. We can extend this to all values of t (i.e. not just integers, but all positive real numbers), and obtain a matrix formally very similar to those of the Jukes–Cantor and Kimura DNA models.

Recall that $S(1)$ was defined by normalising the rows of the symmetric matrix A (p. 42) and then rescaling (to give a 1 PAM matrix). The same result is obtained if these operations are interchanged, rescaling first, then normalising. Since the matrix obtained by rescaling A is symmetric, it can be diagonalised [Mathews & Walker 1970]. Rescaling does not change the diagonal form of the matrix, so we can write $S(1) = UD(\lambda_i)U^{-1}$, where U is a coordinate transformation and $D(\lambda_i)$ is the diagonal matrix with the eigenvalues $\lambda_1 \ldots \lambda_{20}$ down the diagonal. These eigenvalues lie in the range 0 to 1, so can be written $\lambda_i = \exp(-\mu_i)$. Now, the powers of $S(1)$ take a simple form in the diagonal matrix coordinate system; for instance, $S(2) = S(1)S(1) = UD(\lambda_i)U^{-1}UD(\lambda_i)U^{-1} = UD(\lambda_i^2)U^{-1}$, and generally $S(t) = UD(\lambda_i^t)U^{-1}$. Thus we can write

$$S(t) = U \begin{pmatrix} e^{-\mu_1 t} & 0 & \ldots & 0 \\ 0 & e^{-\mu_2 t} & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & e^{-\mu_{20} t} \end{pmatrix} U^{-1}.$$

This shows that each entry of $S(t)$ can be expressed as a sum of exponentials: If A_i denotes the ith amino acid, then $P(A_j|A_i,t) = \sum_k u_{ik}\exp(-\mu_k t)v_{kj}$, where u_{ik} and v_{kj} are the entries in U and U^{-1}, respectively.

This resembles the rate matrices for the DNA models, and we can easily see why. If the Jukes–Cantor rate matrix is diagonalised, so $R = UD(\lambda_i)U^{-1}$ for a suitable coordinate transform U, the equation $S'(t) = S(t)R$ becomes $T'(t) =$

$T(t)D$, where $S(t) = UT(t)U^{-1}$. But the equation $T'(t) = T(t)D$ is easily solved, with the initial condition that $S(0)$ is diagonal; in fact $T(t)$ itself must be diagonal with the terms $\exp(\lambda_i t)$ down its diagonal. Since the eigenvalues are 0 and -4α, it is easy to see that we obtain the Jukes–Cantor matrix entries, (8.4), in a way analogous to the above derivation of the PAM matrices.

Putting $t = \infty$, the PAM matrices become

$$\begin{pmatrix} q_{A_1} & q_{A_2} & \cdots & q_{A_{20}} \\ q_{A_1} & q_{A_2} & \cdots & q_{A_{20}} \\ \cdots & \cdots & \cdots & \cdots \\ q_{A_1} & q_{A_2} & \cdots & q_{A_{20}} \end{pmatrix},$$

where the q_{A_i} are the equilibrium frequencies for amino acids, close to the amino acid frequencies in the database from which Dayhoff, Schwartz & Orcutt [1978] originally constructed their matrices.

Exercise

8.1 Show that the Jukes–Cantor and Kimura substitution matrices are multiplicative.

8.2 Let $P(a(t_2)|b(t_1))$ denote the probability of a residue b, present at time t_1, having been substituted by an a by time t_2. Stationarity means that we can write this as $P(a|b, t_2 - t_1)$. The Markov property means that $P(a(t_2)|b(t_1)) = P(a(t_2)|b(t_1), c(t_0))$ if $t_0 < t_1$, i.e. that the probability of the substitution of b by a is not influenced by the residue being c at the earlier time t_0. Show that

$$\sum_{b(t)} P(a(s+t)|b(t), c(0))P(b(t)|c(0)) = P(a(s+t)|c(0)),$$

and deduce that multiplicativity, (8.1), holds.

8.3 Calculating the likelihood for ungapped alignments

We show here how the likelihood of a tree can be computed using the preceding model. We begin with the case of two sequences, and then proceed to the general case of n sequences.

The case of two sequences

Suppose we have two sequences x^1 and x^2. In this case there is only one tree, namely the one with two branches and a root node which represents the hypothetical common ancestor of x^1 and x^2 (Figure 8.2). Thus we only have to investigate how the likelihood varies with the lengths t_1 and t_2.

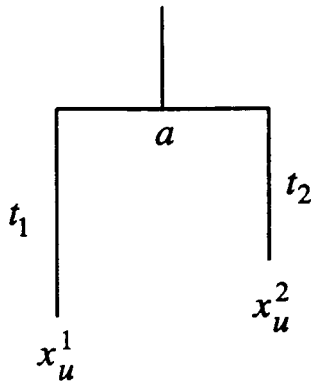

Figure 8.2 *A simple tree.*

Consider a site u. The residues at the leaves 1 and 2 are then x_u^1, x_u^2, respectively. We assign a residue a to the root, using a different notation from the leaves to emphasise the fact that it is a variable, and not specified by the dataset x^1, x^2:

$$P(x_u^1, x_u^2, a|T, t_1, t_2) = q_a P(x_u^1|a, t_1) P(x_u^2|a, t_2).$$

This is the probablity of drawing a from the root distribution (which we assume to be the equilibrium distribution of the substitution matrix family) and of making substitutions of a by x_u^1 and x_u^2. Note that we include the cases where either or both of x_u^1, x_u^2 is the same as a. Since in general we do not know what the root residue was, we must sum over all possible as to get the probability of x_u^1, x_u^2. Formally

$$P(x_u^1, x_u^2|T, t_1, t_2) = \sum_a q_a P(x_u^1|a, t_1) P(x_u^2|a, t_2). \tag{8.7}$$

If there are N sites, we can write the full likelihood as

$$P(x^1, x^2|T, t_1, t_2) = \prod_{u=1}^{N} P(x_u^1, x_u^2|T, t_1, t_2). \tag{8.8}$$

Example: The likelihood of two nucleotide sequences

Suppose we have two nucleotide sequences, and for simplicity, that only two of the nucleotides, C and G, are present. For instance, the sequences might be

CCGGCCGCGCG
CGGGCCGGCCG

What is the likelihood $P(x^1, x^2|T, t_1, t_2)$ of these sequences, assuming the Jukes–Cantor model?

Using the substitution probabilities (8.4), (8.7) gives the probability of C occurring at both leaves of the tree T as:

$$P(\mathrm{C}, \mathrm{C}|T, t_1, t_2) = q_{\mathrm{C}} r_{t_1} r_{t_2} + q_{\mathrm{G}} s_{t_1} s_{t_2} + q_{\mathrm{A}} s_{t_1} s_{t_2} + q_{\mathrm{T}} s_{t_1} s_{t_2} = \tfrac{1}{4}\left(r_{t_1} r_{t_2} + 3 s_{t_1} s_{t_2}\right).$$

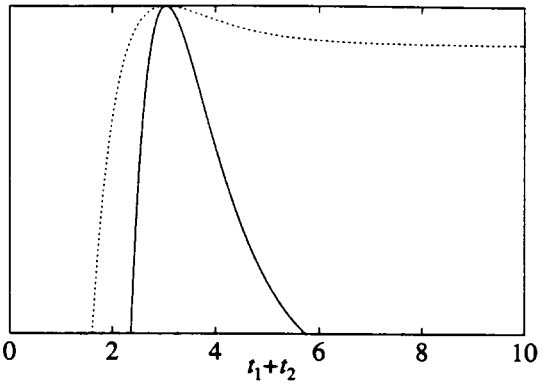

Figure 8.3 *The log likelihood $P(x^1,x^2|T,t_1,t_2)$ given by (8.9), with $n_1 = 100, n_2 = 250$, and with $n_1 = 1000, n_2 = 2500$. The latter curve is sharper, as there are more data to define the maximum likelihood peak. The curves have been shifted so their peaks superimpose.*

By symmetry $P(\mathsf{G},\mathsf{G}|T,t_1,t_2) = P(\mathsf{C},\mathsf{C}|T,t_1,t_2)$. Similarly,

$$P(\mathsf{C},\mathsf{G}|T,t_1,t_2) = P(\mathsf{G},\mathsf{C}|T,t_1,t_2) = \tfrac{1}{4}\left(r_{t_1}s_{t_2} + s_{t_1}r_{t_2} + 2s_{t_1}s_{t_2}\right).$$

Substituting the values r and s gives

$$\begin{aligned} P(\mathsf{C},\mathsf{C}|T,t_1,t_2) &= \tfrac{1}{16}\left(1+3\mathrm{e}^{-4\alpha(t_1+t_2)}\right), \\ P(\mathsf{C},\mathsf{G}|T,t_1,t_2) &= \tfrac{1}{16}\left(1-\mathrm{e}^{-4\alpha(t_1+t_2)}\right). \end{aligned}$$

Now suppose there are n_1 sites where the residues in the two sequences are identical and n_2 sites where a substitution occurs. Then (8.8) gives

$$P(x^1,x^2|T,t_1,t_2) = \frac{1}{16^{n_1+n_2}}\left(1+3\mathrm{e}^{-4\alpha(t_1+t_2)}\right)^{n_1}\left(1-\mathrm{e}^{-4\alpha(t_1+t_2)}\right)^{n_2}. \qquad (8.9)$$

Note that the likelihood depends only on the sum of t_1 and t_2. This is because the Jukes–Cantor substitution process is time-symmetrical, so there is no information available to specify the position of the root. The likelihood remains unchanged if the root slides while the sum $t_1 + t_2$ remains constant. This indeterminacy of the root will be discussed more fully on p. 202. Figure 8.3 shows an example of how the likelihood (plotted as the log likelihood) varies with $t_1 + t_2$. □

The likelihood for an arbitrary number of sequences

We can now extend these calculations to the case of n sequences. Suppose we have a tree T with edge lengths $t_\bullet$. Let $\alpha(i)$ denote the immediate ancestral node to i, i.e. the node at the top of the edge above i. Let $x_u^1 \ldots x_u^n$ denote as usual the residues at the uth site of the n sequences $x^1,\ldots,x^n$. The probability $P(x_u^1 \ldots x_u^n|T,t_\bullet)$ of generating these residues at the n leaves of T is given by

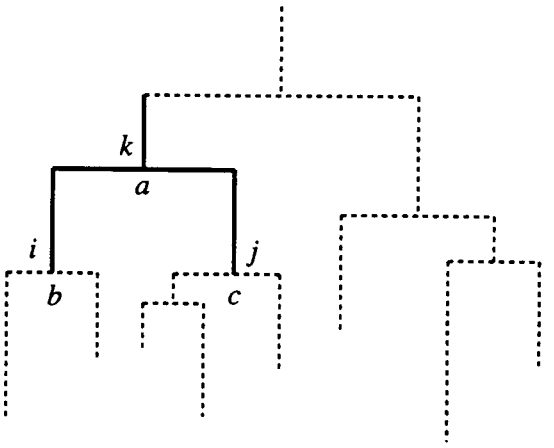

Figure 8.4 *Labelling at a branch in a tree.*

multiplying the probabilities of substitutions at all edges of the tree. Thus

$$P(x_u^1 \ldots x_u^n | T, t_\bullet) = \sum_{a^{n+1}, a^{n+2}, \ldots a^{2n-1}} q_{a^{2n-1}} \prod_{i=n+1}^{2n-2} P(a^i | a^{\alpha(i)}, t_i) \prod_{i=1}^{n} P(x_u^i | a^{\alpha(i)}, t_i) \qquad (8.10)$$

where the sum is over all possible assignments of residues a^k to non-leaf nodes k (these nodes being numbered $n+1$ to $2n-1$).

This probability can be computed by working up the tree from the leaves, using post-order traversal [Felsenstein 1981a]. Let $P(L_k|a)$ denote the probability of all the leaves below node k given that the residue at k is a. Then we compute $P(L_k|a)$ from the probabilities $P(L_i|b)$ and $P(L_j|c)$ for all b and c, where i and j are the daughter nodes of k (Figure 8.4):

Algorithm: Felsenstein's algorithm for likelihood

Initialisation:

Set $k = 2n-1$.

Recursion: Compute $P(L_k|a)$ for all a as follows:

If k is leaf node:

Set $P(L_k|a) = 1$ if $a = x_u^k$, $P(L_k|a) = 0$ if $a \neq x_u^k$.

If k is not a leaf node:

Compute $P(L_i|a)$, $P(L_j|a)$ for all a at the daughter nodes i, j, and set $P(L_k|a) = \sum_{b,c} P(b|a,t_i) P(L_i|b) P(c|a,t_j) P(L_j|c)$.

Termination:

Likelihood at site $u = P(x_u^\bullet | T, t_\bullet) = \sum_a P(L_{2n-1}|a) q_a$. ◁

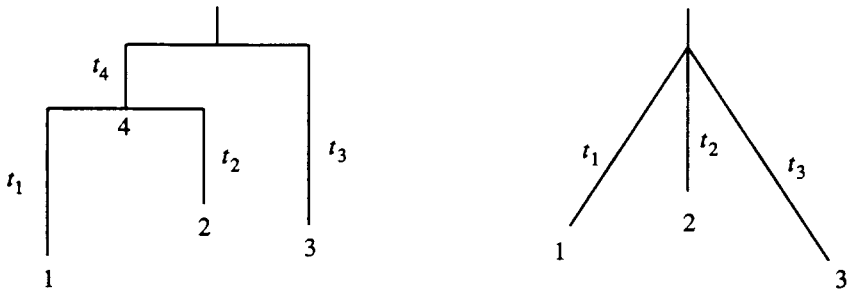

Figure 8.5 *A tree with three leaves (left), which can be simplified to a trifurcating tree (right) for the purpose of computing the likelihood.*

Note the resemblance to the weighted parsimony algorithm (p. 174). We discuss this further on p. 224.

The concluding step in computing the likelihood is to use the assumption of independence at sites to write:

$$P(x^{\bullet}|T,t_{\bullet}) = \prod_{u=1}^{N} P(x_u^{\bullet}|T,t_{\bullet}). \tag{8.11}$$

Example: A tree with three nucleotide sequences

We extend the example on p. 198 now to a tree with three leaves (Figure 8.5, left). The data are three nucleotide sequences composed only of Cs and Gs, for instance:

```
CCGGCCGCGCG
CGGGCCGGCCG
GCCGCCGGGCC
```

We compute the likelihood according to the Jukes–Cantor model. As before, we consider the sites with different assignments of residues separately. Consider the case where C occurs at all leaves. We have

$$\begin{aligned} P(\mathtt{C},\mathtt{C},\mathtt{C}|T,t_1,t_2,t_3) &= q_{\mathtt{C}} r_{t_3}\left(r_{t_4} r_{t_1} r_{t_2} + 3 s_{t_4} s_{t_1} s_{t_2}\right) \\ &\quad + (q_{\mathtt{A}} + q_{\mathtt{G}} + q_{\mathtt{T}}) s_{t_3}\left(r_{t_4} s_{t_1} s_{t_2} + 2 s_{t_4} s_{t_1} s_{t_2} + s_{t_4} r_{t_1} r_{t_2}\right) \\ &= \tfrac{1}{4} r_{t_1} r_{t_2}(r_{t_3} r_{t_4} + 3 s_{t_3} s_{t_4}) \\ &\quad + \tfrac{3}{4} s_{t_1} s_{t_2}(2 s_{t_3} s_{t_4} + s_{t_3} r_{t_4} + s_{t_4} r_{t_3}) \\ &= \tfrac{1}{4}\left(r_{t_1} r_{t_2} r_{t_3+t_4} + 3 s_{t_1} s_{t_2} s_{t_3+t_4}\right), \end{aligned}$$

where the first equation simply computes the terms in (8.10), beginning with the equilibrium probabilities at the root, the second regroups them, and the third follows from the multiplicativity of the Jukes–Cantor matrices (Exercise 8.3). Once again, we find that the lengths of the edges adjoining the root, here t_3 and t_4, appear only as their sum. This holds true for all leaf values, not just 'C, C, C';

it is therefore true for the total likelihood, which allows us to slide the root to node 4, thereby producing a trifurcating tree (Figure 8.5, right). Simplifying the notation by writing t_3 for the third edge (rather than $t_3 + t_4$) we can now compute the likelihood easily, summing over all root assignments and the products of the probabilities of the three edges:

$$P(x_u^1, x_u^2, x_u^3 | T, t_1, t_2, t_3) = \sum_a q_a P(x_u^1 | a, t_1) P(x_u^2 | a, t_2) P(x_u^3 | a, t_3).$$

There are four possible types of terms, where all residues are the same, or where one differs from the other two; for instance:

$$\begin{aligned} P(\mathsf{C}, \mathsf{C}, \mathsf{C} | T, t_1, t_2, t_3) &= \tfrac{1}{4}\left(r_{t_1} r_{t_2} r_{t_3} + s_{t_1} s_{t_2} s_{t_3}\right), \\ P(\mathsf{C}, \mathsf{C}, \mathsf{G} | T, t_1, t_2, t_3) &= \tfrac{1}{4}\left(r_{t_1} r_{t_2} s_{t_3} + s_{t_1} s_{t_2} r_{t_3} + 2 s_{t_1} s_{t_2} s_{t_3}\right). \end{aligned}$$

If there are n_1 sites with the same residue, n_2 of type CCG or GGC, n_3 of type CGC or GCG, and n_4 of type GCC or CGG, then by symmetry

$$\begin{aligned} P(x^1, x^2 | T, t_1, t_2) &= 4^{-3(n_1+n_2+n_3+n_4)} a(t_1, t_2, t_3)^{n_1} b(t_1, t_2, t_3)^{n_2} \\ &\quad \times b(t_1, t_3, t_2)^{n_3} b(t_3, t_2, t_1)^{n_4} \end{aligned} \tag{8.12}$$

where $a(t_1, t_2, t_3)$ and $b(t_1, t_2, t_3)$ are sums of exponentials (see Exercise 8.4). For an illustration of this likelihood function, see Figure 8.6. □

Exercises

8.3 Show that $r_{t_3} r_{t_4} + 3 s_{t_3} s_{t_4}$ and $2 s_{t_3} s_{t_4} + s_{t_3} r_{t_4} + s_{t_4} r_{t_3}$ are terms arising from the product of the Jukes–Cantor matrices for times t_3 and t_4, and deduce that they can be written as $r_{t_3+t_4}$ and $s_{t_3+t_4}$, respectively.

8.4 Show that $a(t_1, t_2, t_3)$ and $b(t_1, t_2, t_3)$ are given by

$$a(t_1, t_2, t_3) = 1 + 3\mathrm{e}^{-4\alpha(t_1+t_2)} + 3\mathrm{e}^{-4\alpha(t_1+t_3)} + 3\mathrm{e}^{-4\alpha(t_2+t_3)} + 6\mathrm{e}^{-4\alpha(t_1+t_2+t_3)}$$

and

$$b(t_1, t_2, t_3) = 1 + 3\mathrm{e}^{-4\alpha(t_1+t_2)} - \mathrm{e}^{-4\alpha(t_1+t_3)} - \mathrm{e}^{-4\alpha(t_2+t_3)} - 2\mathrm{e}^{-4\alpha(t_1+t_2+t_3)}$$

Reversibility and independence of root position

With the parsimony method, we only need to search over unrooted tree topologies. It is much less obvious that the likelihood is independent of the position of the root, but under certain reasonable assumptions this is true. In fact, two assumptions suffice. One is that the substitution matrix family is multiplicative

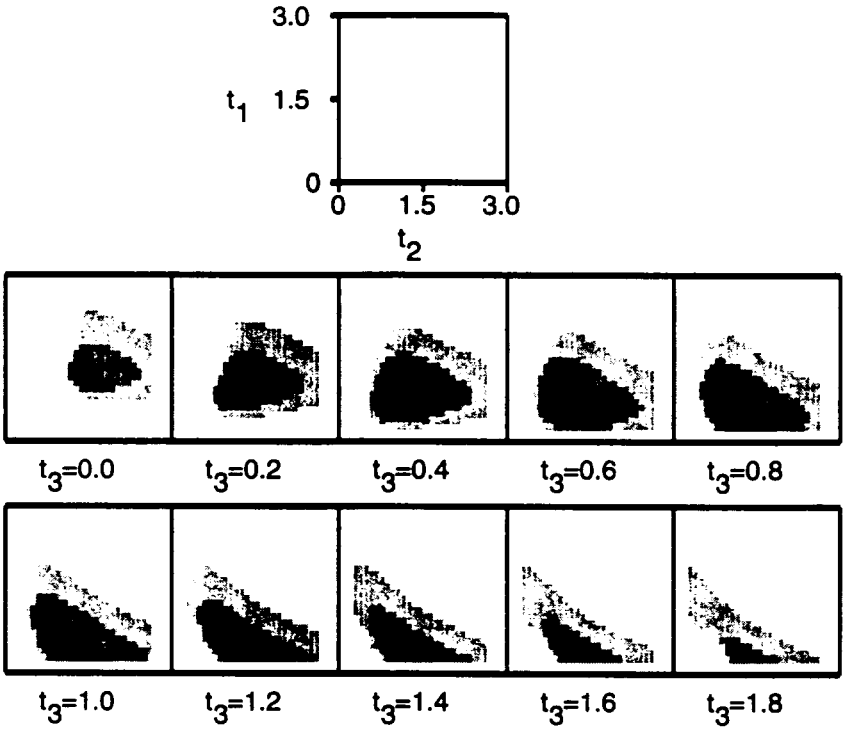

Figure 8.6 *The likelihood function given by (8.12), for* $n_1 = 10$, $n_2 = 20$, $n_3 = 15$, $n_4 = 17$. *White and the five grey levels indicate likelihood values in the ranges separated by* 0, 0.001, 0.01, 0.07, 0.3, 0.9, 1. *Each square shows the likelihood for a particular value of* t_3 *(indicated below the square), the two axes of the square representing* t_1 *and* t_2, *whose ranges are indicated in the square at the top of the figure.*

(8.1), which, as we have seen, holds for the Jukes–Cantor and PAM matrices, amongst others. The other is that *reversibility* should hold. This means that

$$P(b|a,t)q_a = P(a|b,t)q_b \tag{8.13}$$

for all a, b and t. It is clear from their symmetry that reversibility holds for the Jukes–Cantor and Kimura matrices. Reversibility for the PAM matrices follows from the fact that information about the direction of evolutionary time is discarded when the counts are being collected: a substitution from an ancestral residue a to a descendant residue b is treated as equivalent to a substitution in the reverse direction (see Section 2.8 and the example on the PAM family below).

To show that multiplicativity and reversibility imply that all positions of the root give the same likelihood, suppose the two nodes below the root node $2n-1$ are i and j. From the definition of $P(L_{2n-1}|\bullet)$ in terms of $P(L_i|\bullet)$ and $P(L_j|\bullet)$ in Felsenstein's algorithm, we write the likelihood of the sequences $x^\bullet$ at site u as

$$P(x_u^\bullet|T,t_\bullet) = \sum_a q_a P(L_{2n-1}|a) = \sum_{b,c,a} q_a P(b|a,t_i)P(c|a,t_j)P(L_i|b)P(L_j|c),$$

and hence, using reversibility,

$$P(x_u^{\bullet}|T,t_{\bullet}) = \sum_{b,c}\left(\sum_a P(c|a,t_j)P(a|b,t_i)\right) q_b P(L_i|b)P(L_j|c).$$

By multiplicativity, we can rewrite the inner sum as $\sum_a P(c|a,t_j)P(a|b,t_i) = P(c|b,t_i+t_j)$, which means that P is independent of the assignments a of symbols to node $2n-1$, and depends only on the total length of the two edges below the root. Thus the root can be moved freely between i and j, and hence can be moved anywhere within the tree. This is the 'pulley principle' of Felsenstein [1981a]. It implies that the search for the best tree only needs to be carried out on unrooted trees when multiplicative and reversible matrix families are being used.

Example: Reversibility of the PAM family

As remarked in Section 2.8, the counts matrix A used to construct the PAMs is symmetric, i.e. $A_{ab} = A_{ba}$ for all a, b, and since $p_a = \sum_b A_{ab}/\sum_{cd} A_{cd}$, it follows that the normalised matrix B satisfies

$$p_a B_{ab} = A_{ab}/\sum_{cd} A_{cd} = A_{ba}/\sum_{cd} A_{cd} = p_b B_{ba}.$$

Since $S(1)$ is obtained by scaling B, this implies reversibility for $t=1$. To show reversibility of $S(n)$ for all n, suppose we have proved it for all k less than n. Then, applying reversibility for $n-1$ and 1:

$$\begin{aligned} p_a P_n(b|a) &= \textstyle\sum_c p_a P_{n-1}(c|a)P_1(b|c) &&= \displaystyle\sum_c P_{n-1}(a|c)p_c P_1(b|c) \\ &= \textstyle\sum_c P_{n-1}(a|c)P_1(c|b)p_b &&= P_n(a|b)p_b. \end{aligned}$$

□

Example: A non-reversible matrix family

What might a non-reversible matrix look like? Suppose that for two residues A and B the substitution A → B occurs more often than B → A. In order for the frequency of the residues to remain constant, there must be balancing substitutions between other residues. The simplest case is where there are three residues in the alphabet, with a cyclic substitution pattern, giving the instantaneous substitution matrix

$$\begin{pmatrix} -\alpha & \alpha & 0 \\ 0 & -\alpha & \alpha \\ \alpha & 0 & -\alpha \end{pmatrix}. \tag{8.14}$$

This leads to a t-dependent family

$$S(t) = \begin{pmatrix} r_t & s_t & u_t \\ u_t & r_t & s_t \\ s_t & u_t & r_t \end{pmatrix},$$

with

$$\begin{aligned} r_t &= \tfrac{1}{3}\left(1+2\mathrm{e}^{-3\alpha t/2}\cos(\sqrt{3}\alpha t/2)\right), \\ s_t &= \tfrac{1}{3}\left(1-\mathrm{e}^{-3\alpha t/2}\cos(\sqrt{3}\alpha t/2)+\sqrt{3}\mathrm{e}^{-3\alpha t/2}\sin(\sqrt{3}\alpha t/2)\right), \\ u_t &= 1-r_t-s_t. \end{aligned}$$

□

Exercises

8.5 Show that the above family is multiplicative, has positive entries, and substitution rates at $t = 0$ given by (8.14). Find its limiting distribution and show that reversibility, i.e. (8.13), fails for all $t > 0$.

8.6 We have shown that reversibility allows the root to be moved to any position in the tree. What happens when the root is moved to one of the leaf nodes?

8.4 Using the likelihood for inference

We have now reached the heart of probabilistic phylogeny. Having formulated an evolutionary model and defined an algorithm for computing the likelihood from this model, we need to put this machinery to work and infer phylogenetic properties of sets of data. We now give an overview of probabilistic inference methods, beginning with the most venerable and widely used of them, maximum likelihood. We can also use probabilistic methods to assess the quality of the probabilistic model and any variants we devise, but we defer discussing that till we have seen some examples of more elaborate models (see Section 8.5).

Maximising the likelihood

One candidate for the 'best' tree is the tree that maximises the likelihood. Recall that the strategy is to search over trees, and for each topology T to find the lengths $t_\bullet$ that maximise the likelihood $P(x_u^\bullet | T, t_\bullet)$. The topology and the assignment of edge lengths that give the overall maximum of this likelihood is the desired tree.

Given a small number of sequences, say two to five, it is easy to enumerate all trees. For each tree, we can write down the likelihood explicitly as a function of the edge lengths, and maximise it by a suitable numerical technique. This in essence is what Kishino, Miyata & Hasegawa [1990] do, using Newton's method

of optimisation [Press *et al.* 1992]. Their method is intended for maximum likelihood phylogeny of protein sequences, and they use PAM matrices.

For a larger number of sequences, the likelihood can be computed by Felsenstein's algorithm (p. 200). Felsenstein [1981a] also gave an EM algorithm for finding the optimal edge lengths in this case. Alternatively, we can use a standard optimiser, such as conjugate gradients, [Press *et al.* 1992]. This requires the derivatives of the likelihood with respect to the edge lengths, but this is straightforward to compute: we replace $P(y^k|y^{\alpha(k)},t_k)$ wherever it occurs in (8.10) by its derivative $\partial P(y^k|y^{\alpha(k)},t_k)/\partial t_k$.

Even with the best optimiser, maximising the likelihood is computationally demanding, and it is more so for protein sequences because the core computation uses a 20×20 substitution matrix rather than a 4×4 one. To tackle large datasets calls for another strategy; one approach is to use sampling methods.

Exercise

8.7 The maximum likelihood edge lengths can be calculated in certain simple cases. Show that, for our example of two nucleotide sequences (p. 198), the ML solution is given by

$$t_1 + t_2 = -\tfrac{3}{4} \ln \frac{3n_1 - n_2}{3n_1 + 3n_2}.$$

Sampling from the posterior distribution

As we have seen, maximum likelihood is computationally taxing. Furthermore, it is not clear that it is ultimately the best strategy. If we knew the prior $P(T,t_\bullet)$, we could use Bayes' rule to compute the posterior probability $P(T,t_\bullet \mid x^\bullet)$ by

$$P(T,t_\bullet \mid x^\bullet) = \frac{P(x^\bullet|T,t_\bullet)P(T,t_\bullet)}{P(x^\bullet)}.$$

The posterior provides the information we really seek, namely how probable each phylogenetic model is, given the data.

Several authors have used Bayesian methods on small sets of data, where all tree topologies can easily be enumerated (for example Rannala & Yang [1996]). Recently Mau, Newton & Larget [1996] have shown how quite large sequence sets can be handled by sampling from the posterior distribution on the space of trees and edge lengths using the Metropolis algorithm.

To sample from the space of trees is to pick trees randomly with probabilities given by some distribution, in this case their posterior distribution. If we have a large number of samples, then the frequency with which some property of trees is present in the sample converges, in the limit of a large number of samples, to the posterior probability of that property according to the model. For instance, if a

particular tree topology is present in some fraction f of the samples, then f is an estimate for the posterior probability of this topology. We could also determine the probability that a given group is monophyletic, or that one branch point occurs between two others, say, by counting the fraction of cases in which the relevant condition holds. Such questions cannot easily be tackled by likelihood methods, since they require integration over variables, and likelihoods are not probability distributions (see p. 311).

The particular sampling method used by Mau *et al.*, the Metropolis algorithm, is a sampling procedure that generates a sequence of trees, each from the previous one. It assumes that a mechanism is available to generate one tree randomly given another, by sampling from a *proposal* distribution. Let $P_1 = P(T,t_\bullet|x^\bullet)$ be the posterior probability of the current tree, and $P_2 = P(\tilde{T},\tilde{t}_\bullet|x^\bullet)$ that of a proposed new tree. The Metropolis rule is that the new tree is accepted as the next item in the sequence if $P_2 \geq P_1$; if $P_2 < P_1$, the new tree is accepted with probability P_2/P_1. Otherwise the original tree constitutes the next sample (and this repetition of $T,t_\bullet$, if it occurs, is an important part of the process, since we are going to be counting the number of samples with particular properties).[3]

This procedure is guaranteed to sample correctly from the posterior distribution provided that the proposal distribution is symmetrical, in the sense that the probability of proposing $\tilde{T},\tilde{t}_\bullet$ from $T,t_\bullet$ is the same as that of proposing $T,t_\bullet$ from $\tilde{T},\tilde{t}_\bullet$ (see Section 11.4).

Exercise

8.8 Consider a simplified phylogenetic space consisting of two trees T and $\tilde{T}$ with probabilities $P(T)$ and $P(\tilde{T})$. If the proposal procedure always selects the other tree, i.e. the one that is not the current tree, show that the Metropolis algorithm produces a sequence where the frequencies of T and $\tilde{T}$ converge to their probabilities.

A proposal distribution for phylogenetic trees

The choice of the proposal distribution is all-important in making the Metropolis algorithm work well. If the proposed tree is merely randomly selected from the space of all trees, its posterior probability will generally be small, and there will be many wasted repetitions. On the other hand, if the proposed tree is too close to the current tree, many steps will be needed to explore the space of phylogenies adequately. The art lies in finding a way of proposing trees that are promising variants of the current one.

Mau *et al.* suggested a proposal mechanism with two components, one an

[3] Note also that the Metropolis rule only uses the ratio of P_1 and P_2, which is fortunate, because the denominator in Bayes' rule can only be obtained by integrating over the space of trees, and is generally an unknown factor.

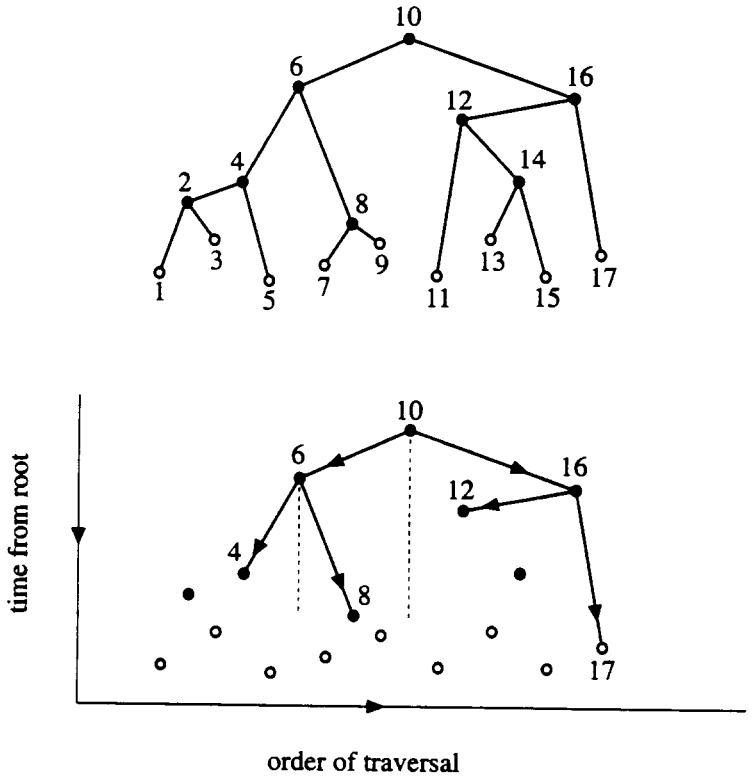

Figure 8.7 *Above: an example of a tree with its nodes numbered in the order of the traversal profile. Below: Reconstruction of the tree from the traversal profile.*

adjustment of lengths of the edges that can also bring about switches in topology in an interesting way, and the other a reordering of the assignment of sequences to leaves. The first uses a representation of a tree that they call a *traversal profile*. This is a diagram completely equivalent to the original tree, but allowing more convenient manipulations of the topology. In the traversal profile,[4] a node is placed at a height corresponding to the sum of the edge lengths from the root to that node. The nodes are regularly spaced horizontally, in the order in which they are encountered during a traversal of the tree. This traversal is defined as follows: Beginning at the leftmost leaf, we traverse the tree depth first from left to right, assigning numbers incrementally, and numbering nodes when we first come to them. This ensures that, for any node with number k, all its left children have numbers lower than k, and all its right children have numbers higher than k. The top diagram in Figure 8.7 shows an example of a tree in which the nodes have been numbered by their order in the traversal profile.

Given a traversal profile, we can reconstruct the tree by a procedure illustrated in Figure 8.7. The root is taken to be the highest node (node 10 in the figure).

[4] In Mau, Newton & Larget [1996], the nodes are connected by lines of constant slope, rather than being equally spaced horizontally, as we have chosen to represent them.

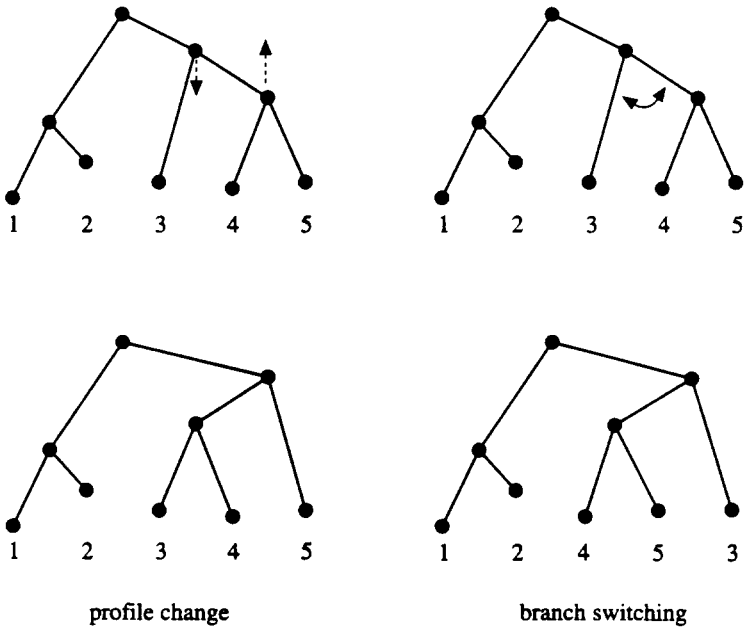

Figure 8.8 *The two parts of the proposal mechanism are changes in the height of the nodes in the profile (left), and reordering of the leaves by switching branches (right). The former can produce changes in the topology, as shown here. The latter does not do this; it just rearranges the existing topology. However, the change of order of the leaves allows new topologies to be reached through further steps of the first type.*

Edges are then drawn to the highest nodes to the left and the right of the root (nodes 6 and 16 in the figure). Suppose now we have reached node k. The daughter nodes of k must lie within the horizontal stretch bounded by any nodes that are higher than k; within this stretch edges are drawn, as before, to the highest nodes to the left and right. Thus, in the figure, the region where the right daughter of node 6 can lie is delimited by the vertical dotted lines. The process stops when a leaf is reached (the leaves have been marked as hollow circles in the figure).

One part of the proposal procedure of Mau *et al.* consists of taking the traversal profile for the current tree and shifting the positions of nodes up and down by an amount chosen from a uniform distribution within certain bounds. Whenever the relative heights of nodes are switched a new topology is produced (Figure 8.8). However, this will never allow leaves which are not adjacent in the traversal order to be neighbours (but see Exercise 8.10). They therefore give an additional proposal mechanism which achieves this. It reorders the leaves by randomly switching the direction of the branches at each node. This produces no change in the posterior probability (so is always accepted), but will lead into a new region of tree space. Adjusting the heights of the traversal profile does of course produce changes in the posterior, but these changes vary continuously with the size of the adjustments, even when there is a change in topology (Exercise 8.9). The

proposal mechanism behaves better in this respect than branch-swopping, which is an intuitively obvious way of modifying trees but has the disadvantage that it is likely to make large changes in the posterior probability.

To define the posterior, a prior has to be chosen over trees. As there is little reliable information available about the distribution of trees, Mau *et al.* assumed a flat prior, assigning equal probabilities to all sets of edge lengths $t_1, \ldots, t_{2n-1}$ for n sequences, for any tree topology. (Note that this does not imply that all tree topologies have equal prior probability; see Exercise 8.11.) To ensure a normalisable probability distribution, they imposed an upper bound on the total edge length from root to any leaf.[5] They found that they could reproducibly identify the most probable topologies for datasets of up to 32 sequences. Their method seems to work best when there is a molecular clock, i.e. when the leaves in the traversal profile are all at the same height.

Exercises

8.9 Consider the profile change shown in the two left-hand figures in Figure 8.8. Suppose the two nodes with arrows in the top figure are at heights h_1 and h_2, and their heights are switched to h_2 and h_1. Show that the resulting change in the likelihood tends to zero as $h_1 - h_2$ tends to zero.

8.10 Show that the two leaves at the extreme ends of the traversal profile can become neighbours, but no other non-adjacent pair can.

8.11 The flat prior on edge lengths assigns a prior to any topology that is obtained by integrating over all possible edge lengths for that topology. This integral will be defined if, following Mau *et al.*, we impose a bound on the total edge length from root to any leaf; call this bound B. Consider the case where there is a molecular clock, and show that the tree with four leaves and topology ((01)(23)) has integrated prior probability $B^3/3$; show that this integral is $B^3/6$ for the topology ((0(12))3). (Hint: Define times from the three branch nodes in each tree to the present, and integrate over these three variables.) This shows that different topologies can have different priors. Show, however, that if one defines a *labelled history* to be a specific ordering for the times of branch nodes relative to present time (assuming a molecular clock), then all labelled histories for four leaves have the same prior probability. Extend this to n leaves.

[5] The limit on edge lengths in the prior might seem an artificial constraint, but it actually has little effect, because trees with extremely long edges generally have low likelihoods. This is because the substitution probabilities over a long edge tend to the equilibrium frequencies, q_a; all correlations with other sequences are therefore neglected.

Other phylogenetic uses of sampling

Sampling methods have not only been used for species or gene phylogeny, but also for inferring the history of populations from a set of present-day individuals. Kuhner, Yamato & Felsenstein [1995] used a sampling method to pick plausible trees, T, relating the individuals of the set. Now, the prior on trees depends on the size of the population, θ. Intuitively, this is because the larger a population is, the further back we expect to go to find the common ancestor of any two individuals. Thus, for each tree T, we can use the prior $P(T|\theta)$ as a likelihood for estimating θ. The sampling process then allows us to accumulate likelihood data in proportion to $P(\text{data}|T)$, giving, in the limit of many samples, the desired likelihood function $\int P(\text{data}|T)P(T|\theta)dT = P(\text{data}|\theta)$.

The proposal mechanism used by Kuhner, Yamato & Felsenstein [1995] is closely related to that of Mau, Newton & Larget [1996]. Instead of adjusting the heights of all nodes in a traversal profile, Kuhner *et al.* adjust the relative heights of *two* nodes, and allow their children to be relabelled. This is therefore a local version of the two components in the method of Mau *et al.*. It would be interesting to know which mechanism samples more effectively.

The prior $P(T|\theta)$ might seem difficult to calculate, since it involves summing over all trees in the population that could provide possible phylogenies for the set of present-day individuals. However, there is a remarkably simple way of evaluating $P(T|\theta)$, based on the idea of running time backwards and allowing branches to *coalesce* [Kingman 1982a; 1982b; Hudson 1990]. For a fixed, large, population size, the probability density of a coalescence in time turns out to be $2/\theta$. Imagine a horizontal line rising through the tree T, starting from the level of the leaves. Each time a coalescence occurs, the number of edges will fall by one. Suppose the time between the coalescence from k to $(k-1)$ edges is τ_k. Then the probability of a coalescence in the interval dt between two of the $k(k-1)/2$ pairs of edges is $k(k-1)dt/\theta$, so $(2/\theta)\exp(-\tau_k k(k-1)/\theta)$ is the probability of coalescence occurring at the end of the period τ_k and not before. Taking the product over all intervals τ_k gives the total probability of the tree

$$P(T|\theta) = \left(\frac{2}{\theta}\right)^{n-1} \exp\left(-\sum_{k=2}^{k=n} \frac{k(k-1)\tau_k}{\theta}\right).$$

Closely related to the coalescent is a prior obtained by a simple evolutionary model, where we think of a tree as being formed by a series of splitting events. If there is a constant probability density, λ say, of a split occurring in a growing edge, the splitting is said to follow a *Yule process*. The resulting prior on trees has a simple form, being proportional to $\exp(-\lambda \sum t_i)$, for edge lengths t_i (see Exercise 8.12). This is different from the coalescent prior because it assumes that all the descendants of the root sequence are present at the leaves, without omissions or extinctions whereas the coalescent prior treats the species or genes

as being picked randomly from a large pool. Which prior is more appropriate depends on whether a taxonomist is looking at a small closely related family or a wide-ranging selection.

Exercises

8.12 Under a Yule process, the probability density for no split occurring during the interval 0 to t is given by the limit of $(1-\lambda\delta t)^{t/\delta t}$ as $\delta t \to 0$, and is therefore $\exp(-\lambda t)$. Deduce that the Yule prior for a tree with n leaves is proportional to $\exp(-\lambda \sum t_i)$, where the t_i are all edge lengths. Following the same reasoning as in Exercise 8.11, show that the priors for all labelled histories on four leaves are equal under the Yule prior. Extend this to the case of n leaves.

8.13 Assuming a molecular clock, calculate the expected lengths of all the branches of rooted trees with two, three or four leaves under the Yule prior with splitting rate λ and the coalescent prior with population size θ. (Hint: Consider the case of three leaves. Let the two short edges have length s and the long edge length t. The total edge length of the tree is then $2t+s$, so the tree probability for the Yule process is proportional to $\exp(-\lambda(2t+s))$. Check that the coalescent probability for the tree is proportional to $\exp(-(2t+4s)/\theta)$. Now compute the means of s and t for these distributions in the standard manner, integrating over all $0 \le t \le \infty$ and $0 \le s \le t$.)

The bootstrap revisited

The bootstrap, p. 179, can be applied to maximum likelihood, just as to other tree building methods. Artificial data are generated by drawing columns randomly with replacement from the true dataset, and then the maximum likelihood tree is found for the artificial dataset. The frequency of occurrence of some feature over many replications of this procedure measures the confidence we have in inferring it by maximum likelihood.

One therefore obtains information similar to that obtained by sampling from the posterior, and in fact the two methods are related: For some phylogenetic models, the bootstrap confidence for a feature approximates the posterior probability of that feature, assuming a flat prior over trees. To gain some intuition for why this is so, consider the simple case of maximum likelihood estimation of the probability of getting a head in a coin toss, on the basis of a set of data.

Example: Bootstrapping the results of a coin toss experiment

A coin is tossed N times, giving m heads (H) and n tails (T). The posterior distribution for the probability p of a head, assuming a flat prior, is given by the

Dirichlet distribution

$$P(p|m\mathrm{H},n\mathrm{T}) = p^m(1-p)^n\frac{(N+1)!}{m!n!}. \tag{8.15}$$

A bootstrap trial starts by drawing a set of N coin tosses from the data, with probability m/N of H, n/N of T. If there are k heads in this set, the maximum likelihood probability estimated from the set is $p^{ML} = k/N$. Thus

$$P\left(p^{ML} = k/N\right) = \left(\frac{m}{N}\right)^k\left(\frac{n}{N}\right)^{N-k}\binom{N}{k}. \tag{8.16}$$

For large N, we can approximate this by the distribution

$$P\left(p^{ML} = p\right) = \left(\frac{m}{N}\right)^{Np}\left(\frac{n}{N}\right)^{N-Np}(N+1)\binom{N}{Np} \tag{8.17}$$

where the factor $(N+1)$ appears because we have replaced $(N+1)$ terms of the binomial expansion with a density on $[0,1]$. For large N we can approximate (8.15) by a normal distribution (p. 300):

$$P(p|m\mathrm{H},n\mathrm{T}) \simeq \frac{(N+1)}{\sqrt{2\pi Np(1-p)}}\exp\left(-\frac{(m-Np)^2}{2Np(1-p)}\right), \tag{8.18}$$

and similarly (8.17) becomes

$$P\left(p^{ML} = p\right) \simeq \frac{(N+1)}{\sqrt{2\pi mn/N}}\exp\left(-\frac{(m-Np)^2}{2mn/N}\right). \tag{8.19}$$

It is a straightforward exercise to show that, if N is large enough, i.e. if there is plenty of data, these two distributions approximate one another closely (Exercise 8.14). □

This result can easily be extended to multinomial distributions. In phylogenetic examples, the probability of drawing particular leaf assignments to make the bootstrap dataset is governed by a multinomial distribution. We consider next a case where this distribution collapses to a binomial, and the coin tossing example above can be directly applied.

Example: The bootstrap for a simple tree

Suppose we have two nucleotide sequences which we wish to model using a tree with two leaves and the Jukes–Cantor substitution matrices. We can place the root at one of the leaves, and set t to be the length of the edge connecting the two leaves. Let n_s denote the number of sites in the original data set for which the two leaves take the same nucleotide, and let n_d be the number of sites where they differ; if there are N sites altogether, $n_d + n_s = N$. Extending the calculation in (8.9), and assuming a flat prior on $\mathrm{e}^{-\alpha t}$, we can write the posterior probability as

$$P(\mathrm{e}^{-\alpha t}|\mathrm{data}) = (1+3\mathrm{e}^{-\alpha t})^{n_s}(1-\mathrm{e}^{-\alpha t})^{n_d}/Z, \tag{8.20}$$

where Z is a normalising factor from Bayes' theorem. Suppose n_{XY} denotes the number of leaf assignments of type XY in the original dataset, and m_{XY} the corresponding number in a bootstrap dataset. The probability of drawing the bootstrap dataset is given by the multinomial distribution

$$P(m_\bullet|n_\bullet) = \left(\frac{n_{AA}}{N}\right)^{m_{AA}} \left(\frac{n_{AC}}{N}\right)^{m_{AC}} \cdots \frac{N!}{m_{AA}!m_{AC}!\ldots}. \tag{8.21}$$

Now the maximum of the likelihood for the bootstrap dataset is given by an obvious extension of Exercise 8.7 as

$$\exp(-\alpha t^{ML}) = \frac{3m_s - m_d}{3N}, \tag{8.22}$$

where m_s, m_d are the number of equal and differing leaves in the bootstrap set. Thus t^{ML} depends only on m_s and m_d, and not on the individual counts m_{XY}, and all $P(m_\bullet|n_\bullet)$s given by (8.21) that have the same m_s and m_d can be summed in determining the frequency with which the bootstrap value t^{ML} will occur. Summing over these terms gives

$$P\left(e^{-\alpha t^{ML}} = \frac{3m_s - m_d}{3N}\right) = \left(\frac{n_s}{N}\right)^{m_s} \left(\frac{n_d}{N}\right)^{m_d} \binom{N}{m_s}. \tag{8.23}$$

Comparing (8.20) with (8.15) and (8.16) with (8.23), and noting that (8.22) implies $m_s = N(1+3\exp(-\alpha t^{ML}))/4$ and $m_d = 3N(1-\exp(-\alpha t^{ML}))/4$, we see that the posterior for this tree approximates the bootstrap, just as for coin tossing. □

Thus the bootstrap distribution can, for certain phylogenetic models, and with enough data (large enough N), give a good approximation to the posterior. However, the labour involved in evaluating a large number of maximum likelihood trees makes this use of the bootstrap an unattractive alternative to sampling. The bootstrap is probably more useful for non-probabilitistic tree building methods. However, the relationship between the bootstrap and the posterior does give some insight. In particular, it helps to counter objections raised by Hillis & Bull [1993]. These authors generated sample datasets from a tree and found the distribution of the frequency with which a given tree topology was reconstructed by parsimony; for each sample dataset they also obtained the bootstrap frequency of that topology. They found the distribution of bootstrap frequencies was far wider than that of the original samples. This is unsurprising, however, since sampling followed by bootstrapping adds variance from two steps. In fact, the bootstrap distribution in their simulations has the correct variance *given the data* [Efron, Halloran & Holmes 1996], i.e. when viewed as a posterior distribution.

Exercise

8.14 Show that, for sufficiently large N, $P(p|m\mathrm{H},n\mathrm{T})$ given by (8.18) and $P(p^{ML}=p)$ given by (8.19) are either both very small or else take nearly equal values.

8.5 Towards more realistic evolutionary models

The evolutionary models used so far have made some fairly drastic simplifying assumptions (p. 193). The restriction to ungapped alignments discards useful phylogenetic information given by the pattern of deletions and insertions. It is also clearly incorrect to model each site in a sequence with the same substitution matrix, as assumed in (8.11), since there are different constraints at different sites, imposed by structure of proteins, base pairing of RNA, and so on. To focus on a single basic property of sites, it has long been known that substitutions occur much more rapidly at some sites than others [Fitch & Margoliash 1967b]. We describe first some attempts to model this behaviour by allowing variable rates of evolution, and then turn to ways of treating gapped alignments.

Allowing different rates at different sites

The basic strategy of maximum likelihood is to pick a tree T and a set of lengths $t_\bullet$ and compute the likelihood over all sites, using

$$P(x^\bullet|T,t_\bullet)=\prod_{u=1}^{N}P(x_u^\bullet|T,t_\bullet).$$

Yang [1993] suggested introducing a site-dependent variable, r_u, that scales all the $t_\bullet$ at the site u. If we knew the values of r_u at each site, we could write the likelihood as

$$P(x^\bullet|T,t_\bullet,r_u)=\prod_{u=1}^{N}P(x_u^\bullet|T,r_u t_\bullet).$$

Since we generally do not know the values of r_u, our best strategy is to assume a prior for them, and integrate over all values of each r. Yang [1993] used a gamma distribution $g(r,\alpha,\alpha)$ as his prior; this has mean 1 and variance $1/\alpha$, and therefore allows a range from a tight distribution (α large) to a broad one (α small). The likelihood is

$$P(x^\bullet|T,t_\bullet,\alpha)=\prod_{u=1}^{N}\int_0^\infty P(x_u^\bullet|T,rt_\bullet)g(r,\alpha,\alpha)dr. \qquad (8.24)$$

For each fixed T, this likelihood is maximised with respect to $t_\bullet$ and α. Using a set of globins, Yang derived an optimal tree for four mammals, and showed