5

Phylogeny Estimation

> When I took up my post in Pavia in October 1961, [...] Cavalli-Sforza introduced me to his idea of developing methods for a computer-based construction of evolutionary trees from contemporary data [...]. Initially, I was skeptical. I had studied linkage estimation theory under Fisher himself and I knew how difficult it was. I had visions of evolutionary tree estimation being much the same but with the addition of the need to estimate the form of the tree itself, surely a fatal complexity: my intuition was that there would be insufficient data for the task. Cavalli-Sforza was very persuasive, however.
>
> —Edwards [63]

Reconstructing the evolutionary relationships among living species is one of the oldest problems in biology. There have been some real advances during the past two decades, but several difficulties remain.

- The estimation of phylogenies is a computationally hard problem which is analytically intractable in the general case [43].
- Realistic models of character evolution involve many parameters, and it is likely that real processes are much more complex than the most complex models available in the literature.
- A common biological complication is that the species and the characters under study do not have the same history; this is particularly the case for genetic data [12].
- It is often necessary to estimate many parameters simultaneously though only some of them are of interest [135].
- There is some confusion in the use of some terminology related to estimation and statistics that is likely to reveal difficulties in communicating across different scientific fields [138].
- Some confusion arises because phylogeny estimation methods are also used for systematics (i.e., classification of species) rather than estimating evolutionary parameters.

E. Paradis, *Analysis of Phylogenetics and Evolution with R*, Use R,
DOI 10.1007/978-1-4614-1743-9_5, © Springer Science+Business Media, LLC 2012

- Many studies assessed the "performance" of phylogenetic methods using simulations but these considered only special cases, and the conclusions drawn from these simulations are of very limited value [138].
- The different methods, models, and algorithms for phylogeny estimation are available in distinct programs resulting in several practical difficulties [169, 203, 202].

The last point is of particular interest here. All these programs have their own features and requirements in terms of operating systems, user interfaces, data formats, or licenses. Comparing different methods is difficult because it is often hard to decide whether the observed differences in the results are due to different assumptions, algorithms, run-time environments, computer architectures, or other features that vary among programs. Even the analysis of a single data set is made difficult by the need to switch between different software and / or operating systems.

All phylogeny estimation methods are characterized by:

1. A numerical criterion to measure the adequacy of a given tree topology compared to another.
2. A formula to calculate the branch lengths.
3. An algorithm to explore the tree space.

The development of phylogeny estimation in R is very recent, and significant progress has been made in distance-based and maximum likelihood methods. This is limited compared to the methods available in the literature (particularly with respect to the old, well-established parsimony methods, and the current success of Bayesian methods). There are good reasons to focus on distance and likelihood methods, because these methods have been shown to perform well in a number of situations (although we have to be cautious in generalizing these conclusions as mentioned above). There has been a long-lasting debate on the merits of parsimony, and although this method has been severely criticized [77], it can be viewed as a valid nonparametric method [138]. Bayesian methods enjoy a current success, but some critics pointed out the limitations of this approach [79, 293]. However, Bayesian phylogeny estimation may be implemented in a straightforward way because all the necessary ingredients exist in R or have been developed in various packages as we will see later in this chapter.

5.1 Distance Methods

Distance methods have a long history because they are generally tractable even with a large amount of data [298]. Consider a phylogenetic tree $\mathcal{T}$ among a set of species. $\mathcal{T}$ defines a unique matrix of pairwise distances, Δ, given by the paths between species (the patristic distances):

$$\mathcal{T} \longrightarrow \Delta$$

However, for a given distance matrix, many trees may be defined depending on the criterion used to define them:

$$\Delta \longrightarrow \mathcal{T}_1, \mathcal{T}_2, \ldots$$

All the problematic of distance-based methods is to find the tree for a given distance matrix. The driving idea is that the distances in Δ give information on the relative closeness of observations. The distances are used to estimate both the topology of the tree and its branch lengths. There are two main strategies to find a tree: aggregating the most closely related observations, or splitting the most distant sets of observations. We will see below that for a given data set (sequences, measurements, ...) there are several possible distance matrices depending on the method used to compute them.

There has been considerable progress in implementing these methods in R, particularly based on the numerous methods to compute distances in R from various types of data which are the focus of the first section.

5.1.1 Calculating Distances

There is a difference between the concepts of statistical and evolutionary distances. In statistics, a distance can be viewed as a "physical" or geometric distance between two observations, each variable being a dimension in a hyperspace. In evolutionary biology, a distance is an estimate of the divergence between two units (individuals, populations, or species). This is usually measured in quantity of evolutionary change (e.g., numbers of mutations).

R has various functions to compute distances available in different packages. Table 5.1 lists these functions, which are detailed in the following sections.

Classical Distances

There are several ways to define a distance between two observations using numerical data. It would be too long to detail all of them here. Fortunately, the help pages of the functions mentioned below include the formulae of the different methods.

`dist` in package `stats` performs distance calculations taking a matrix as its main argument. Its main option is `method` which can take one of the six following strings: `"euclidean"` (the default), `"maximum"`, `"manhattan"`, `"canberra"`, `"binary"`, or `"minkowski"`. As a simple example, consider three variables taken on three individuals:

```
> X <- matrix(c(0, 1, 5), 3, 3)
> rownames(X) <- LETTERS[1:3]
> X
```

Table 5.1. Functions for computing distances in R

Package	Function	Data Types
stats	`dist`	Continous or binary
	`cophenetic`	Objects of class `"hclust"` or `"dendrogram"`
cluster	`daisy`	Continuous and / or discrete
ade4	`dist.binary`	Binary
	`dist.prop`	Relative frequencies
	`dist.quant`	Continous
ape	`dist.gene`	Discrete
	`dist.dna`	Aligned DNA sequences
	`weight.taxo`	'Taxonomic' levels
	`cophenetic`	An object of class `"phylo"`
adephylo	`distTips`	Objects of class `"phylo"`, `"phylo4"` or `"phylo4d"`
adegenet	`dist.genpop`[a]	An object of class `"genpop"`
phangorn	`dist.ml`	An alignment of amino acids

[a]Replaces `dist.genet` from ade4

```
  [,1] [,2] [,3]
A    0    0    0
B    1    1    1
C    5    5    5
```

These may be viewed as three points in a 3-d space where A would be at the origin of the space.

```
> dist(X)
          A         B
B 1.732051
C 8.660254 6.928203
> dist(X, method = "maximum")
  A B
B 1
C 5 4
> dist(X, method = "manhattan")
   A  B
B  3
C 15 12
> dist(X, "binary")
  A B
B 1
C 1 0
```

`dist` returns an object of class `"dist"` which is a vector storing only the lower triangle of the distance matrix (because it is symmetric and all its diagonal elements are equal to zero). This class can be converted into a matrix using the generic function `as.matrix`, and a matrix can be converted with `as.dist`:

```
> d <- dist(X)
> class(d)
[1] "dist"
> as.matrix(d)
          A        B        C
A 0.000000 1.732051 8.660254
B 1.732051 0.000000 6.928203
C 8.660254 6.928203 0.000000
```

The function `daisy` in the package cluster also performs distance calculations but it implements some methods that can deal with mixed data types. Three metrics are available via the option `metric`: `"euclidean"`, `"manhattan"`, or `"gower"`. The last one implements Gower's coefficient of similarity for mixed data types [112]. The types of the variables are either identified with respect to the class of the columns, or specified with the option `type` (see `?daisy` for details). In the example below, we convert the columns of `X` as factors to build the data frame `Y`:

```
> daisy(X, "gower")
Dissimilarities :
    A   B
B 0.2
C 1.0 0.8

Metric :  mixed ;  Types = I, I, I
Number of objects : 3
> Y <- as.data.frame(apply(X, 2, factor))
> Y
  V1 V2 V3
A  0  0  0
B  1  1  1
C  5  5  5
> daisy(Y, "gower")
Dissimilarities :
  A B
B 1
C 1 1

Metric :  mixed ;  Types = N, N, N
Number of objects : 3
```

`dist.quant` in ade4 implements three metrics (canonical or Euclidean, Joreskog, and Mahalanobis) specified with an integer between 1 and 3:

```
> dist.quant(X, 1) # canonical
          A        B
```

```
B 1.732051
C 8.660254 6.928203
> dist.quant(X, 2) # Joreskog
          A         B
B 0.8017837
C 4.0089186 3.2071349
> dist.quant(X, 3) # Mahalanobis
          A         B
B 0.4629100
C 2.3145502 1.8516402
```

`dist` and `daisy` handle missing data and will usually correct the distances for their presence, but the returned distance may be `NA` if there are too many missing values. On the other hand, `dist.quant` does not accept missing values.

Evolutionary Distances

`dist.gene` provides a simple interface to compute the distance between two haplotypes using a simple binomial distribution of the pairwise differences. This allows one to compute easily the variance of the estimated distances using the expected variance of the binomial distribution. The input data are a matrix or a data frame where each row represents a haplotype, and each column a locus. Missing values are accepted and handled through the option `pairwise.deletion`. By default, the columns with at least one `NA` are deleted before computing (global deletion), so that all distances are computed with the same columns. If a few missing values are spread over all columns, this procedure may result in very few (even none) variables left for computing the distances. If `pairwise.deletion = TRUE`, columns are deleted on a pairwise basis and the distances are adjusted with the respective numbers of variables used.

`dist.dna` provides a comprehensive function for the estimation of distances from aligned DNA sequences using substitution models or counts of the number of differences (`"n"`, `"raw"` scales by the sequence length), transitions (`"ts"`), transversions (`"tv"`), or alignement gaps (`"indel"`, `"indelblock"` counts contiguous gaps as a single unit). The options are detaild in Table 5.2. The substitution models are described in Section 5.2.1.[1] If a correction for among-sites heterogeneity (usually based on a Γ distribution) is available, this may be taken into account. The variances of the distances can be computed as well. Missing and ambiguous nucleotides are taken into account with the option `pairwise.deletion` as described in the previous paragraph.

`dist.genpop` takes as input an object of class `"genpop"`. Five methods are available to compute these distances: standard (or Nei), angular (or Edwards),

[1] `dist.dna` calculates distances for the models whose an analytical formula has been found. See page 144 for a general numerical formula.

Table 5.2. Options of the function `dist.dna`. The values marked with (d) are the default ones

Options	Effect	Possible Values
`model`	Specifies the substitution model	`"raw"`, `"n"`, `"ts"`, `"tv"`, `"indel"`, `"indelblock"`, `"JC69"`, `"K80"` (d), `"K81"`, `"F81"`, `"F84"`, `"BH87"`, `"T92"`, `"TN93"`, `"GG95"`, `"logdet"`, `"paralin"`
`variance`	Whether to compute the variances	`FALSE` (d), `TRUE`
`gamma`	The value of α for the Γ correction	`NULL` (no correction) (d), a numeric giving the value of α
`pairwise.deletion`	Whether to delete the sites with missing data in a pairwise way	`FALSE` (d), `TRUE`
`base.freq`	The frequencies of the four bases	`NULL` (calculated from the data) (d), four numeric values
`as.matrix`	Whether to return the results as a matrix or as an object of class `"dist"`	`TRUE` (d), `FALSE`

Reynolds, Rogers, and Provesti. This is specified with the option `method` which takes an integer value between 1 and 5.

These three functions return an object of class `"dist"`, except `dist.dna` with `model = "BH87"` because the Barry–Hartigan model is asymmetric [21].

Special Distances

The package ade4 has two functions that compute distances with some special types of data: `dist.binary` and `dist.prop`, for binary data and proportions, respectively. The first one has the option `method` which takes an integer between 1 and 10; this includes the well-known Jaccard, and the Sokal and Sneath methods. The second function has a similar option taking an integer between 1 and 5: this includes Rogers's, Nei's, and Edwards's methods.

ape has the function `weight.taxo` that computes a similarity matrix between observations characterized by categories that can be interpreted as a taxonomic level (i.e., a numeric code, a character string, or a factor). The value is 1 if both observations are identical, 0 otherwise.

stats has a generic function `cophenetic` that computes the distances among the tips of a hierarchical data structure: there are methods for objects of class `"hclust"`, `"dendrogram"`, and `"phylo"`. Additionally, adephylo has the function `distTips` computing various distances from a tree (patristic, node path, Abouheif's, and sum of descendants of nodes on path). ade4 and vegan have a comprehensive list of ecological distances.

5.1.2 Exploring and Assessing Distances

The choice of a particular distance is often dictated by the type of data at hand, but in most situations it is very useful to compare different distances computed with the same data. Because distances are easily computed, they may be potentially useful exploratory tools that have been underexploited in phylogenetic analyses. It is important to emphasize that even if a distance-based method is not used, pairwise distances contain information that could be explored before tree estimation.

A simple graphical exploration of the distances may be revealing. Because an object of class `"dist"` (say `d`) contains only the lower triangle of the distance matrix, it can be plotted with `hist(d)`. It is well-known that the way classes are defined in a histogram may hide patterns in the data, so it is useful to compare the default number of classes used by `hist` (approximately 20) with, for instance, `hist(d, 50)`. An alternative is to draw a curve of the empirical density on top of the histogram plotted as densities, instead of numbers, so its total area equals one:

```
hist(d, freq = FALSE)
lines(density(d))
```

lattice gives a nice alternative with `densityplot(~ d)`. Ideally, with the objective of phylogeny estimation in mind, one wants to have the distances spread as much as possible in order to obtain a resolved phylogeny. One has to be careful on the scale of the x-axis here because similarly shaped histograms may have different scales. Small distances may indicate observations that will be difficult to pull apart in the phylogeny (typically, < 0.005 for a sequence of 1000 sites). On the other hand, large distances will lead to a similar problem because of mutation saturation: the JC69 distance cannot exceed 0.75, but in practice distances greater than 0.3 lead to difficulties.

A second useful, though very simple, graphical exploration tool is to plot two sets of distances computed on the same data with the goal to contrast the effect of different methods. A form of this approach is well-known as the 'saturation plot' where the number of transitions or transversions is plotted against the K80 distance (see p. 191), but this can be generalized. For instance, the F81 model differs from the JC69 one by considering unbalanced base frequencies (p. 142). So plotting these two distances together will show the impact of this assumption:

```
djc69 <- dist.dna(x, "JC69")
dk81 <- dist.dna(x, "K81")
plot(djc69, dk81)
abline(b = 1, a = 0)
```

We take advantage of the fact that the observations are ordered in the same way in both `"dist"` objects. In case of doubt this may be checked with:

```
all(attr(djc69, "Labels") == attr(dk81, "Labels"))
```

Distance matrices may be characterized by a number of properties that may give helpful indications on the evolutionary information in the data. We will consider only a few of these properties. A particularly interesting one is *additivity*. A distance matrix is said to be additive if it satisfies the *four-point condition*. Consider a quartet of observations $q = \{x, y, u, v\}$, and define:

$$d_{xy|uv} = d_{xy} + d_{uv}.$$

Then the four-point condition is satisfied for q if among the three quantities $d_{xy|uv}$, $d_{xu|yv}$, and $d_{xv|yu}$ the two largest ones are equal. Additive distances may be represented appropriately with a tree structure. Holland et al. [136] developed an exploratory method to assess the addivity of a distance matrix. They define the ratio:

$$\delta_q = \frac{d_{xv|yu} - d_{xu|yv}}{d_{xv|yu} - d_{xy|uv}},$$

under the condition that $d_{xy|uv} \leq d_{xu|yv} \leq d_{xv|yu}$. If the distances are additive δ_q will be close to zero. The function `delta.plot` implements this method. δ_q is computed for all possible quartets, and their distribution is examined with a histogram. Additionally, a mean value $\bar{\delta}$ is computed for each observation taking all δ_q values where this observation is involved. For illustration, we take a distance matrix computed from a set of fifteen sequences of cytochrome b, and another matrix computed from a random set of five variables for fifteen observations (Fig. 5.1):

```
> data(woodmouse)
> dw <- dist.dna(woodmouse)
> x <- replicate(5, rnorm(15))
> dx <- dist(x)
> delta.plot(dw)
> delta.plot(dx)
```

A difficulty with δ-plots is that the shape of the distribution of the δ_q depends on several factors, including the number of variables (e.g., if `x` had only one column in the above example, then $\delta_q = 0$ for all quartets). It may thus be appropriate to examine some hypothetical scenarios with simulated data.

Another relevant property is whether a distance is *Euclidean*. Euclidean distances are measured in a purely isotropic space, i.e., a space where classical geometry applies. Most multivariate methods, such as principal components analysis, requires variables to be represented in a Euclidean space so that rotations are meaningful. `ade4` has the function `is.euclid` that returns `TRUE` or `FALSE`. Note that for a given data set, a distance matrix may or may not be Euclidean depending on the method used. A non-Euclidean distance matrix can be transformed into a Euclidean one by multiplying the distances

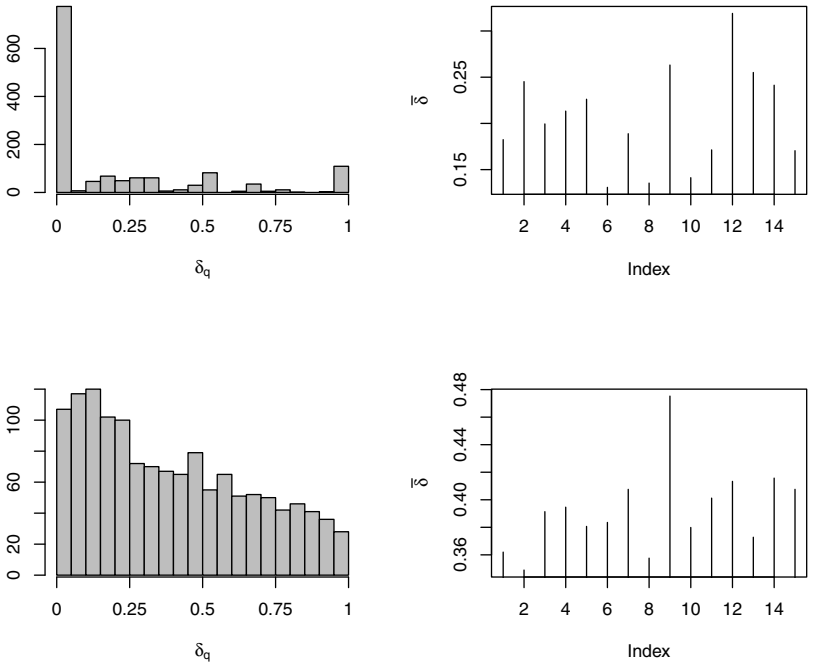

Fig. 5.1. δ-plot of fifteen sequences of cytochrome b (top) and fifteen random observations (bottom)

by a coefficient that is found from an eigen decomposition of the original matrix. Two functions are available in `ade4`: `cailliez` that does a simple linear transformation of the distances, and `lingoes` that does the same on the squared distances. Note also the function `quasieuclid` to 'clean' an already Euclidean matrix which is not completely so because of numerical approximations (Thibaut Jombart, personal communication). Recently, de Vienne, Aguileta and Ollier [51] showed that taking the square root of the distances make them Euclidean. This simple transformation, done with `sqrt(d)`, implies less distortion of the original distances compared to the Caillez transform.

A well-known property of distances in evolutionary biology is *ultrametricity*. A distance matrix is ultrametric if it satisfies the following inequality:

$$d_{xy} \leq \max(d_{xz}, d_{yz}) \ ,$$

for all triplets $\{x, y, z\}$. Consider the case where there are only three observations: it is clear that the above condition is met only if all distances are equal. So if we represent these three observations with a tree, it would have

three equal branch lengths, and all tips would be equally distant from the root which is the definition of an ultrametric tree. Ultrametricity is appealing for evolutionists because if evolution is homogeneous across lineages—but not necessarily through time—then distances among them are expected to be ultrametric. Several decades of research have shown that this will be not generally true with real data. Fortunately, a distance matrix may be non-ultrametric but still be additive. The reader is invited to explore further these ideas in the Exercices of this chapter.

Campbell, Legendre and Lapointe [36, 37] developed a test of the congruence among ultrametric distance matrices. It is implemented in the functions `CADM.global` and `CADM.post` in `ape`.

5.1.3 Simple Clustering, UPGMA, and WPGMA

There is a corpus of phylogeny estimation methods that are based on statistical clustering methods. They were popular in the past, but have recently declined since the rise of likelihood and Bayesian methods. These methods are aggregative. The first step is to find the two observations with the shortest pairwise distance in the matrix: this distance is used to calculate the 'height' at which they aggregate, that is the distance from the node to each of these tips. Since no other observation come into this height calculation, it is not possible to assume different rates of change in both branches, so they have the same length. The second step is to calculate the distances from the inferred node to the remaining observations: this is where clustering methods differ because there are many ways to recalculate distances. Once this is done, the two steps are repeated until no observation remains.

R has a reasonably large number of functions that perform clustering [307]. They mostly work on a distance (also called dissimilarity) matrix, but some of them work directly on the original data matrix (observations and variables). The unweighted pair-group method using arithmetic average (UPGMA) is similar to a hierarchical clustering with the average method as implemented in the `hclust` function. The only difference is that the branch lengths are halved in the UPGMA method. This is implemented in the function `upgma` in `phangorn`, for instance:

```
M <- dist.dna(woodmouse)
tr <- upgma(M)
```

The substitution model can be changed with the appropriate option in `dist.dna`. Giving the graphical functions detailed in the previous chapter, it is easy to compare the trees estimated with different substitution models. For instance:

```
M1 <- dist.dna(woodmouse) # K80 is the default
tr1 <- upgma(M1)
M2 <- dist.dna(woodmouse, model = "F84")
```

```
tr2 <- upgma(M2)
layout(matrix(1:2, 2, 1))
plot(tr1, main = "Kimura (80) distances")
plot(tr2, main = "Felsenstein (84) distances")
```

We show some practical examples in Section 5.8.

`upgma` has an option `method` that specifies the method used to update the distances in the second step described above. The default is `"average"`; six other methods are available and described in the help page `?hclust`. Setting `method = "mcquitty"` is the same thing that using the function `wpgma` that implements the weighted version or WPGMA. The topology and branch lengths of the tree are substantially affected by the choice of this method (see Exercices).

5.1.4 Neighbor-Joining

The neighbor-joining (NJ) is one of the most widely used methods of phylogeny estimation. It is a splitting method. The first step is to build a tree with a single internal branch where one node is linked to two observations (the neighbors), and the other is linked to all the others. All possible pairs of neighbors are considered: the tree with the smallest total branch length is selected. The second step is to update the distance matrix by removing the two neighbors and calculating the distance from the new node to the remaining observations. The two steps are repeated until the tree is dichotomous. The branch lengths are estimated by least squares, so the rates of evolution may vary among branches.

`ape` has the function `nj` that performs the NJ algorithm. Its use is simple: it takes a distance matrix as unique argument, and returns the estimated tree as an object of class `"phylo"`. As for the UPGMA, it is easy to obtain NJ trees with different substitution models. It is also possible to call `nj` repeatedly for a series of models:

```
mod <- list("JC69", "K80", "F81", "F84")
lapply(mod, function(m) nj(dist.dna(X, model = m)))
```

In the above command, we insert the call to `dist.dna` with the call to `nj` in a function where the model is treated as a variable. `lapply` then dispatches the different models to this function, and returns the results as a list.

Because branch lengths are estimated by least squares, it happens sometimes that some of them may be negative. This is the case with the woodmouse data:

```
> trw <- nj(dist.dna(woodmouse))
> which(trw$edge.length < 0)
[1] 5
> trw$edge.length[5]
[1] -3.160774e-05
```

Usually, these negative branch lengths are tiny (the mean branch length of `trw` is 2.43×10^{-3}), and it seems reasonable to consider them as equal to zero if needed in subsequent analyses [102, 172].

The properties of NJ have been extensively studied in the literature [e.g., 100, 103, 273]. Particularly, NJ has good statistical properties when distances are unbiased.

5.1.5 Extensions of Neighbor-Joining: UNJ and BIONJ

We have seen above that clustering methods differ essentially in the way the distance matrix is updated after two observations have been clustered. Similarly, the NJ method may vary with respect to the way distances are updated after a splitting step. In the original version of the method, the new distances are calculated with:

$$d_{ui} = \frac{d_{xi} + d_{yi} - d_{xy}}{2} \qquad i \neq x, y \ ,$$

where x and y are the two neighbors, and u is the new node to be added in the matrix. During the first iteration of the NJ algorithm, x and y are original observations (species, populations, ...) but in the subsequent iterations they can be nodes created during the previous steps and so representing more than one observations. The unweighted neighbor-joining (UNJ) takes this into account by correcting the above formula with the number of observations (i.e., tips) in x and y [99]. It is unweighted in the sense that all observations are given the same weight [101]. The UNJ method is implemented in the function `UNJ` in phangorn.

So far, we have ignored the variances and covariances of the distances which are considered as independent and with equal variance. This is unlikely to be generally true because of the shared evolutionary history which makes distances non-independent (a theme we will revisit extensively in the next chapter), and the longer distances will tend to have higher variances. However, computing the variances and covariances of the distances is computationally complicated. Gascuel [98] proposed an approximation in the case where distances are from molecular sequences: the variances are then calculated with the distances read in the matrix divided by the sequence length. These approximations are used to calculate d_{ui} in a way to minimize the variance of the distances. This is the BIONJ method, and implemented in the function `bionj` in `ape`.

5.1.6 Minimum Evolution

Desper and Gascuel [55] developed a method that aims to alleviate some of the limitations of NJ. Because their method is based on the principle of minimum evolution, and it is faster than others, the authors named it FastME. They developed two versions of it. In the OLS (ordinary least squares) version, a

tree is first built agglomeratively using a fast algorithm. In a second step, a method of tree rearrangements based on nearest-neighbors interchange (NNI; Fig. 5.2) is applied until the shortest tree has been found. These two steps do not require the calculation of the branch lengths—in contrast to NJ which does so at each iteration. Finally, branch lengths are estimated on the final tree by least squares. This method is implemented in the function `fastme.ols` in `ape`:

```
fastme.ols(X, nni = TRUE)
```

The second version of FastME is based on a "balanced" approach to computing distances. This uses a formula established by Pauplin [236] to calculate the length of a tree without requiring to calculate its branch lengths. The procedure is otherwise close to the OLS version. However, the balanced approach allows to use more complex tree rearrangements, namely subtree pruning and regrafting (SPR), and tree bisection and reconnection (TBR).[2] This is implemented in `fastme.bal` also in `ape`:

```
fastme.bal(X, nni = TRUE, spr = TRUE, tbr = TRUE)
```

Desper and Gascuel wrote that FastME is faster than NJ [55]. In practice with the present implementation, however, `fastme.ols` is faster than `nj` while `fastme.bal` is slower, particularly if TBR rearrangements are used. Nevertheless, the overall computing time of the latter remains reasonable even with a large data set (≈ 15 s with 500 observations on a modern laptop). On the other hand, Desper and Gascuel claimed a greater accuracy of the balanced version of FastME compared to NJ, which should make this method an attractive alternative to NJ with moderate-sized data sets.

We have seen repeatedly the importance of least squares in distance methods. These may be formally defined with:

$$\sum_{i<j}(d_{ij} - t_{ij})^2 \,, \tag{5.1}$$

where i and j are the observations, d_{ij} is as before, and t_{ij} is the distance from the tree. This quantity is used for estimation in several methods, but it can also be used as a measure of the goodness-of-fit of the estimated tree. The generic function `cophenetic` returns the distances from a tree: it is then possible to compare them with the original distances. For instance, taking the woodmouse data and estimating an NJ tree:

```
d <- dist.dna(woodmouse)
tr.nj <- nj(d)
dt.nj <- cophenetic(tr.nj)
```

[2] At the time of writing of this book, these improvements are not yet published in the literature (Olivier Gascuel, personal communication).

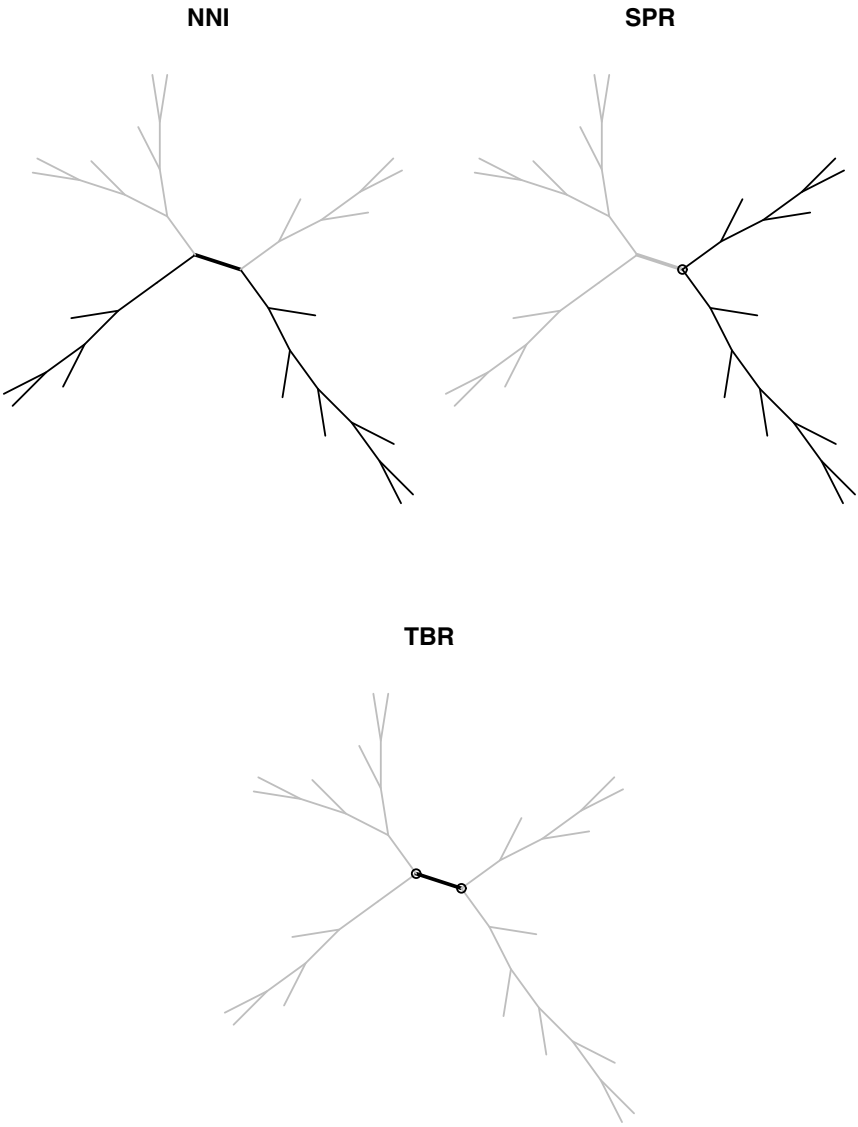

Fig. 5.2. The three common tree rearrangement operations consider an internal branch (in thick line). NNI: the two grey subtrees are exchanged; there are two possible NNIs for each internal branch. SPR: the grey subtree is moved to an internal branch of the black subtree; the circled node is deleted and a new one is created where the grey subtree is regrafted. TBR: both grey subtrees are moved in the same way than in SPR; the two circled nodes are deleted and a new one is created on each subtree

We have to insure that the distances are ordered in the same way in `d` and in `dt.nj`. The latter is a matrix—not an object of class `"dist"`—which helps to reorder its rows and columns:

```
dmat <- as.matrix(d)
nms <- rownames(dmat)
dt.nj <- dt.nj[nms, nms]
dt.nj <- as.dist(dt.nj)
```

We have converted `dt.nj` as a `"dist"` object to have each distance only once and remove the diagonal of zeros. There are several ways to plot these two sets of distances, for instance:

```
plot(dt.nj - d, ylab = "distance residuals")
abline(h = 0, lty = 3)
```

to display something similar to a plot of residuals in a regression analysis. This is done using four tree estimation methods in Fig. 5.3. UPGMA shows a clear larger dispersion of these 'residuals' compared to the three other methods. Interestingly, BIONJ shows a more compact dispersion around the $y = 0$ line of most points, though the extreme points are similar to NJ and FastME.

Note that regression residuals are assumed to be independent; however, we cannot make the same assumption for the points in Fig. 5.3 because they are calculated using pairwise distances.

It is simple from the above code lines to build functions to ease this kind of goodness-of-fit diagnostics, or even to propose other kinds of plots. The reader is invited to explore these ideas in the Exercices at the end of this chapter.

5.2 Maximum Likelihood Methods

Maximum likelihood is the cornerstone of modern statistics [61, 67]. Applying this estimation method to phylogenetic problems is simple in principle. Suppose we have data on a trait from three species, one mammal, one bird, and one fish. With a statistical model of the evolution of this trait and the phylogeny ((mammal,bird),fish), it is possible to compute the probabilities of the trait values on the tree. There are two big difficulties though. First, choosing a model of evolution seems to have a particular impact on the result of phylogeny estimation which raises the question of what is a "good" model of trait evolution. Second, it is rare that we know the tree of the species or sequences under study, and since this is the question of interest to most researchers this has attracted considerable attention.

Before diving in these two issues, the first section gives the bases of substitution models as applied to molecular sequences which are the most widely used data for phylogeny estimation.

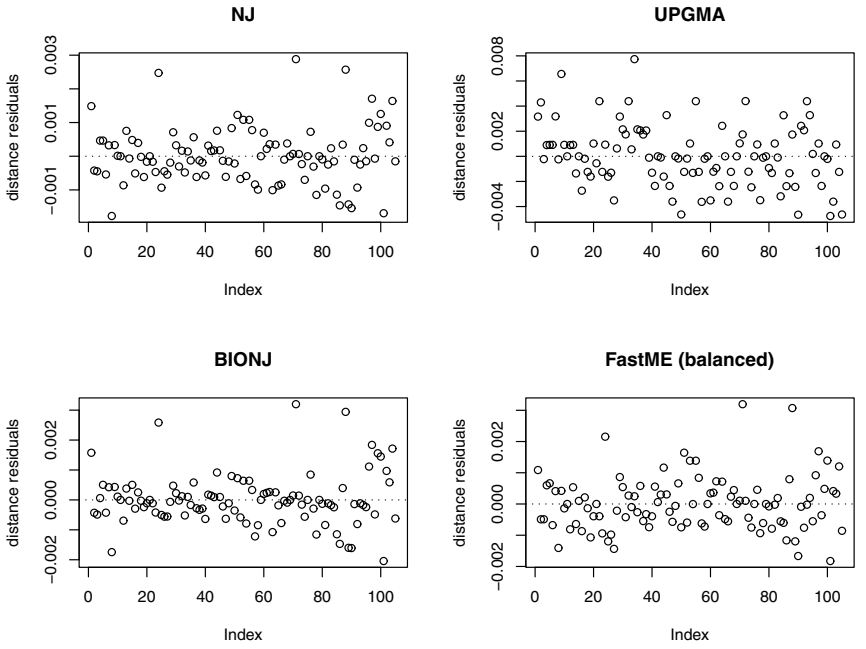

Fig. 5.3. Plot of the difference between the distances from the estimated tree and original distances (distance residuals) for four tree estimation methods

5.2.1 Substitution Models: A Primer

The vast majority of models of evolution for discrete characters are Markovian implying that:

- The number of character states is finite;
- The probabilities of transitions among these states are controlled by some parameters;
- The process is at equilibrium.

This can be applied to many kinds of data, but the recent rise of large-scale molecular databases has led to this approach being applied essentially to nucleotide (DNA) and amino acid (protein) sequences.

A substitution model is a formulation of the instantaneous rate(s) of change among the different states of the character. For instance, for a character with two states, A and B, where the rate of change (i.e., the probability of change from one state to another for a very short time) is symmetric and equal to 0.1, the *rate matrix*, usually denoted Q, is:

$$Q = \begin{bmatrix} -0.1 & 0.1 \\ 0.1 & -0.1 \end{bmatrix} . \tag{5.2}$$

The rows of Q correspond to the initial state, and its columns to the final one. The elements on the diagonal are set so that the sum of each row is zero. For an arbitrary time interval t, the *probability matrix* P is obtained by the matrix exponentiation of Q:

$$P = \mathrm{e}^{tQ} \ . \tag{5.3}$$

The element p_{ij} from the ith row and jth column of P is the probability of being in state j after time t given that the initial state was i. The probabilities in P take into account possible multiple changes (e.g., a change from A to B may be the result of A $\rightarrow$ B, or A $\rightarrow$ B $\rightarrow$ A $\rightarrow$ B, ...). The matrix exponentiation is usually calculated with an infinite sum:

$$\mathrm{e}^{tQ} = I + tQ + \frac{(tQ)^2}{2!} + \frac{(tQ)^3}{3!} + \cdots \tag{5.4}$$

$$= I + \sum_{i=1}^{\infty} \frac{(tQ)^i}{i!} \ . \tag{5.5}$$

In practice, an approximation is done. `ape` has the function `matexpo`:

```
> Q <- matrix(c(-0.1, 0.1, 0.1, -0.1), 2)
> Q
     [,1] [,2]
[1,] -0.1  0.1
[2,]  0.1 -0.1
> matexpo(Q) # t = 1
           [,1]       [,2]
[1,] 0.90936538 0.09063462
[2,] 0.09063462 0.90936538
> matexpo(10*Q) # t = 10
          [,1]      [,2]
[1,] 0.5676676 0.4323324
[2,] 0.4323324 0.5676676
```

We effectively have probabilities because the rows sum to one. Note that Q is independent of time whereas P is not. Both calculated matrices are symmetric; they could be asymmetric if Q were.

With longer time t, the probabilities in P tend to be equal. Indeed if $t = 1000$, all probabilities would be equal to 0.5 in the above example. This is an important property of Markovian models: the final state is independent of the initial state and its probabilities are equal to the equilibrium frequencies of the states. Here we assumed that these frequencies are balanced, but it does not need to be always true: in that case, the transition rates must be multiplied by these frequencies. For instance, if the equilibrium frequencies of A and B are 0.95 and 0.05, Q and P become:

```
> Q <- matrix(c(-0.005, 0.095, 0.005, -0.095), 2)
> Q
       [,1]   [,2]
[1,] -0.005  0.005
[2,]  0.095 -0.095
> matexpo(Q) # t = 1
           [,1]        [,2]
[1,] 0.99524187 0.004758129
[2,] 0.09040445 0.909595547
> matexpo(Q*1000) # t = 1000
     [,1] [,2]
[1,] 0.95 0.05
[2,] 0.95 0.05
```

When fitting a substitution model to some data, its parameter(s) will usually be unknown. For the hypothetical two-state character we write:

$$Q = \begin{bmatrix} \cdot & \alpha \\ \alpha & \cdot \end{bmatrix}, \tag{5.6}$$

where α is the parameter and the dots on the diagonal indicate that these values are set so that the rows sum to zero. There may be other parameters, for instance, if the equilibrium frequencies are to be estimated as well, there would be one additional parameter in Q:

$$Q = \begin{bmatrix} \cdot & (1 - f_A)\alpha \\ f_A\alpha & \cdot \end{bmatrix}.$$

This is generalized to DNA sequences (by assuming that Q is 4×4), or to protein sequences (20×20). The substitution models differ in the way the rate matrix Q is modeled. We consider here in detail the case of DNA sequences because substitution models for this kind of data are implemented in several functions in various packages.

For the simplest models of DNA substitution, it is possible to derive the transition probabilities (i.e., the elements of P) without matrix exponentiation: this is nicely explained by Felsenstein [79, p. 156]. Substitution models are used to calculate distances between pairs of sequences (p. 128) in which case the parameters may not be estimated [317]. When these models are used in phylogeny estimation, their parameters are estimated from the data (see details below). We focus on the structure of the models and give pointers to the literature for the formulae (e.g., [317] for a thorough treatment).

Jukes and Cantor 1969 (JC69)

This is the simplest model of DNA substitution [155]. The probability of change from one nucleotide to any other is the same. It is assumed that all four bases have the same frequencies (0.25). The rate matrix Q is:

$$\begin{array}{c} \\ \mathrm{A} \\ \mathrm{G} \\ \mathrm{C} \\ \mathrm{T} \end{array} \begin{array}{c} \begin{array}{cccc} \mathrm{A} & \mathrm{G} & \mathrm{C} & \mathrm{T} \end{array} \\ \begin{bmatrix} . & \alpha & \alpha & \alpha \\ \alpha & . & \alpha & \alpha \\ \alpha & \alpha & . & \alpha \\ \alpha & \alpha & \alpha & . \end{bmatrix} \end{array} .$$

As with the general case above, the rows correspond to the original state of the nucleotide, and the columns to the final state (the row and column labels are omitted in the following models).

The overall rate of change in this model is thus 3α. The probability of change from one base to another during time t can easily be derived (see [79]):

$$p_{ab}(t) = (1 - \mathrm{e}^{-4\alpha t})/4 \qquad a \neq b \,, \tag{5.7}$$

where a and b are among A, G, C, and T.

The expected mean number of substitutions between two sequences is $3(1 - \mathrm{e}^{-4\alpha t})/4$ because there are three different types of change. From this, it is straightforward to derive an estimate of the distance ($\hat{t}$).

Kimura 1980 (K80)

Because there are two kinds of bases with different chemical structures, purines (A and G) and pyrimidines (C and T), it is likely that the changes within and between these kinds are different. Kimura [161] developed a model whose rate matrix is:

$$\begin{bmatrix} . & \alpha & \beta & \beta \\ \alpha & . & \beta & \beta \\ \beta & \beta & . & \alpha \\ \beta & \beta & \alpha & . \end{bmatrix} .$$

A change within a type of base is called a *transition* and occurs at rate α; a change between types is called a *transversion* and occurs at rate β. The base frequencies are assumed to be equal.

Felsenstein 1981 (F81)

Felsenstein [74] extended the JC69 model by relaxing the assumption of equal frequencies. Thus the rate parameters are proportional to the latter:

$$\begin{bmatrix} . & \alpha\pi_G & \alpha\pi_C & \alpha\pi_T \\ \alpha\pi_A & . & \alpha\pi_C & \alpha\pi_T \\ \alpha\pi_A & \alpha\pi_G & . & \alpha\pi_T \\ \alpha\pi_A & \alpha\pi_G & \alpha\pi_C & . \end{bmatrix} .$$

There are three additional parameters (the base frequencies, π_A, π_G, π_C, and π_T, sum to one, thus only three of them must be estimated) but they are usually estimated from the pooled sample of sequences.

Kimura 1981 (K81)

Kimura [162] generalized his model K80 by assuming that two kinds of transversions have different rates: A $\leftrightarrow$ C and G $\leftrightarrow$ T on one side, and A $\leftrightarrow$ T and C $\leftrightarrow$ G on the other.

$$\begin{bmatrix} . & \alpha & \beta & \gamma \\ \alpha & . & \gamma & \beta \\ \beta & \gamma & . & \alpha \\ \gamma & \beta & \alpha & . \end{bmatrix}.$$

Felsenstein 1984 (F84)

This model can be viewed as a synthesis of K80 and F81: there are different rates for base transitions and transversions, and the base frequencies are not assumed to be equal. The rate matrix is:

$$\begin{bmatrix} . & \pi_G(\alpha/\pi_R+\beta) & \beta\pi_C & \beta\pi_T \\ \pi_A(\alpha/\pi_R+\beta) & . & \beta\pi_C & \beta\pi_T \\ \beta\pi_A & \beta\pi_G & . & \pi_T(\alpha/\pi_Y+\beta) \\ \beta\pi_A & \beta\pi_G & \pi_C(\alpha/\pi_Y+\beta) & . \end{bmatrix},$$

where $\pi_R = \pi_A + \pi_G$, and $\pi_Y = \pi_C + \pi_T$ (the proportions of purines and pyrimidines, respectively). Felsenstein and Churchill [83] gave formulae for the probability matrix and the distance.

Hasegawa, Kishino, and Yano 1985 (HKY85)

This model is very close in essence to the previous one but its parameterization is different [129]:

$$\begin{bmatrix} . & \alpha\pi_G & \beta\pi_C & \beta\pi_T \\ \alpha\pi_A & . & \beta\pi_C & \beta\pi_T \\ \beta\pi_A & \beta\pi_G & . & \alpha\pi_T \\ \beta\pi_A & \beta\pi_G & \alpha\pi_C & . \end{bmatrix}.$$

Due to some mathematical properties of this rate matrix, it is not possible to derive analytical formulae of the transition probabilities, and so for the distance as well [314].

Tamura 1992 (T92)

The model developed by Tamura [296] is a generalization of K80 that takes into account the content of G + C. The rate matrix is:

$$\begin{bmatrix} . & \alpha\theta & \beta\theta & \beta(1-\theta) \\ \alpha(1-\theta) & . & \beta\theta & \beta(1-\theta) \\ \beta(1-\theta) & \beta\theta & . & \alpha(1-\theta) \\ \beta(1-\theta) & \beta\theta & \alpha\theta & . \end{bmatrix},$$

where $\theta = \pi_G + \pi_C$. Tamura [296] gave formulae for the distance, and Galtier and Gouy [94] gave formulae for the transition probabilities.

Tamura and Nei 1993 (TN93)

Tamura and Nei [297] developed a model where both kinds of base transitions, A $\leftrightarrow$ G and C $\leftrightarrow$ T, have different rates α_R and α_Y, respectively. The base frequencies may be unequal. All the above models can be seen as particular cases of the TN93 model. The rate matrix is:

$$\begin{bmatrix} . & \pi_G(\alpha_R/\pi_R + \beta) & \beta\pi_C & \beta\pi_T \\ \pi_A(\alpha_R/\pi_R + \beta) & . & \beta\pi_C & \beta\pi_T \\ \beta\pi_A & \beta\pi_G & . & \pi_T(\alpha_Y/\pi_Y + \beta) \\ \beta\pi_A & \beta\pi_G & \pi_C(\alpha_Y/\pi_Y + \beta) & . \end{bmatrix}.$$

Fixing $\alpha_R = \alpha_Y$ results in the F84 model, whereas fixing $\alpha_R/\pi_R = \alpha_Y/\pi_Y$ results in the HKY85 model [79].

The General Time-Reversible Model (GTR)

This is the most general time-reversible model. All substitution rates are different, and the base frequencies may be unequal [176]. The rate matrix is:

$$\begin{bmatrix} . & \alpha\pi_G & \beta\pi_C & \gamma\pi_T \\ \alpha\pi_A & . & \delta\pi_C & \epsilon\pi_T \\ \beta\pi_A & \delta\pi_G & . & \zeta\pi_T \\ \gamma\pi_A & \epsilon\pi_G & \zeta\pi_C & . \end{bmatrix}.$$

There are no analytical formulae for the transition probabilities, nor for the distance. Nonetheless, it is possible to calculate a distance with [317]:

$$\hat{t} = -\text{trace}\{\hat{\Pi} \ln(\hat{\Pi}^{-1}\hat{F})\}, \tag{5.8}$$

where $\hat{\Pi}$ is a diagonal matrix with the observed base frequencies, $\hat{F}$ is a 4×4 matrix with the observed relative frequencies of pairs of nucleotides, and 'ln' is the matrix logarithm calculated as $\ln(X) = VUV^{-1}$ with V the matrix of

eigenvalues of X and U a diagonal matrix with the logarithm of the eigenvalues of X. This is not implemented in a phylogenetic package, but it can computed using standard R commands and this is the subject of an exercise at the end of this chapter.

It appears from the structure of the GTR model in the above matrix that a number of intermediate models, albeit still time-reversible, can be defined by constraining the six substitution rates $(\alpha, \dots)$ and / or the four base frequencies $(\pi_A, \dots)$. phangorn implements twenty three of these models in its function `modelTest`: these include six of the models described above (F84 and T92 being not included in phangorn's scheme; Table 5.4, p. 152).

Barry and Hartigan 1987, LogDet, and Paralinear

More complex models than the GTR can be built by removing the constraint of time-reversibility: the most general model would thus have twelve substitution rates—plus eventually the three frequency parameters. The models though are difficult to analyze with real data, and their general usefulness in data anlysis is not clear [79, 317].[3]

Barry and Hartigan developed a formula to compute the distance from a general model [21]. They claim this distance is valid even when the model is non-stationary (i.e., the rates are not assumed to be constant through time). The distance matrix is not symmetric, for instance examing the six first columns and rows with the woodmouse data:

```
> round(dist.dna(woodmouse, "BH87")[1:6, 1:6], 4)
         No305  No304  No306 No0906S No0908S No0909S
No305   0.0000 0.0174 0.0165  0.0230  0.0182  0.0179
No304   0.0136 0.0000 0.0041  0.0169  0.0119  0.0155
No306   0.0125 0.0039 0.0000  0.0127  0.0077  0.0113
No0906S 0.0148 0.0125 0.0085  0.0000  0.0101  0.0137
No0908S 0.0158 0.0132 0.0092  0.0158  0.0000  0.0146
No0909S 0.0160 0.0175 0.0135  0.0201  0.0152  0.0000
```

The level of asymmetry is visible to the third digit. This clearly leads to the virtual impossibility to use the Barry–Hartigan distance in distance-based methods. Lake [174] and Lockhart et al. [186] derived variants of the Barry–Hartigan distance that are symmetric: the LogDet and paralinear distances, respectively. These three distances are available in `dist.dna`.

Galtier and Gouy 1995

Galtier and Gouy [93] developed a nonequilibrium model where the G + C content is allowed to change through time. Sequences are assumed to evolve

[3] Rodríguez et al. [269] showed that models that are more complex than GTR do not have an estimable distance with pairwise data.

on each lineage depending on its G + C content. This is estimated from the G + C content of the recent species or populations. It is thus necessary to estimate ancestral G + C contents. The rate matrices for each lineage are similar to the one for the T92 model except that θ may vary. This model is available in `dist.dna`.

Amino Acid Models

Substitution models applied to amino acid sequences use a radically different approach than the one used for nucleodite data: they are based on empirical rate matrices and amino acid frequencies derived from databases of protein sequences so that the rate matrix Q is given, and no parameter has to be estimated. Various authors established different versions of Q; nine of them are available in phangorn for phylogeny estimation (Table 5.3). Additionally, the function `dist.ml` computes distances for aligned amino acid sequences under these models.

Table 5.3. Models of amino acid sequence evolution available in phangorn

Model	Description	Ref.
Dayhoff	General model developed in the late 1970's	[50]
JTT	Improved version of the above	[153]
WAG	General improved model with a maximum likelihood approach	[311]
LG	Improved version of the above	[178]
cpREV	Model for amino acid sequences from chloroplast genes	[4]
mtmam	Model for mitochondrial proteins of mammals	[318]
mtArt	Model for proteins of arthropods	[1]
MtZoa	General model for mitochondrial proteins of animals	[271]
mtREV24	General model developed in the late 1990's	[3]

5.2.2 Estimation with Molecular Sequences

If the probabilities of change along a tree are known (using one of the models described in the previous section), the likelihood of the tree can be computed. However, the states of the data on the nodes of the tree are unknown, and it is necessary to sum the probabilities for all possible states on the nodes which may involve a very large number of terms even for a moderate data set. Felsenstein [74] presented an algorithm that allows considerable time saving in this computation. The idea is to compute successively the likelihoods of each character state at each node by summing the probabilities giving the likelihoods of the descendants (hence the name "pruning algorithm").

Denote as M the number of states (e.g., $M = 4$ for DNA data), $p_{ab}(t)$ the probability of change from state a to state b during time t. Then the likelihood

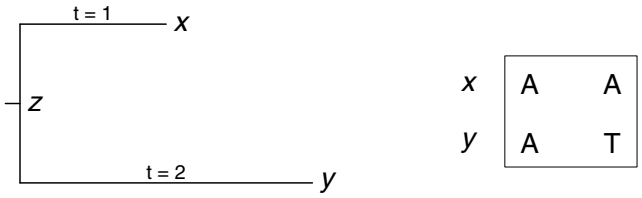

Fig. 5.4. A tree with tips x and y (left) and a matrix with two nucleotides (right)

of state a at node z, given the likelihood of its descendants x and y (assuming a binary tree) and the branch lengths t_{xz} and t_{yz} is:

$$L_{az} = \left(\sum_{b=1}^{M} p_{ab}(t_{xz})L_{bx}\right)\left(\sum_{b=1}^{M} p_{ab}(t_{yz})L_{by}\right) . \tag{5.9}$$

If x is a tip, then $L_{bx} = 1$ if state b is observed, 0 otherwise. This equation looks complicated but its logic is apparent with a very simple example. Figure 5.4 shows a tree (or a part of a bigger tree) with two tips, x and y, where two nucleotides have been observed: the first site is the same for both tips (A) while the second site is polymorphic. To make things simple, we shall use the Jukes–Cantor model with $\alpha = 0.05$. So we only have to use (5.7) to build the probability matrices; for the branch between x and z with $t_{xz} = 1$:

```
> pab <- (1 - exp(-4 * 0.05))/4
> Px <- matrix(pab, 4, 4)
> diag(Px) <- 1 - 3 * pab
> Px
          [,1]       [,2]       [,3]       [,4]
[1,] 0.86404806 0.04531731 0.04531731 0.04531731
[2,] 0.04531731 0.86404806 0.04531731 0.04531731
[3,] 0.04531731 0.04531731 0.86404806 0.04531731
[4,] 0.04531731 0.04531731 0.04531731 0.86404806
```

The calculations are the same for the branch between y and z except that $t_{yz} = 2$: this creates the matrix `Py` (not shown). Consider the first site: the datum is the same for x and y, so we create two identical vectors with the appropriate 1 and 0's:

```
> Lx <- Ly <- c(1, 0, 0, 0)
```

We can compute (5.9) now. Below are two ways to do this: the first command, using a `for` loop, has the advantage of displaying clearly its relationship with the mathematical equation, but the second one, using R recycling rule, is more efficient from all points of view:

```
> # for (i in 1:4) print(sum(Px[i, ]*Lx) * sum(Py[i, ]*Ly))
> colSums(Px * Lx) * colSums(Py * Ly)
[1] 0.650403570 0.003735052 0.003735052 0.003735052
```

Not surprisingly, the likelihood of z having A on this first site is the highest, but the likelihoods of the other states are not zero.

Now consider the second site. The datum is the same for x, so we change only `Ly`. The probability matrices are unchanged (the branch lengths and α are the same), so calculation proceeds exactly as before:

```
> Ly <- c(0, 0, 0, 1)
> colSums(Px * Lx) * colSums(Py * Ly)
[1] 0.071214832 0.003735052 0.003735052 0.034112155
```

Further opportunities to discuss these calculations are provided in the Exercises.

Once this computation has been applied to all nodes of the tree, the likelihood of the character for the tree is obtained by:

$$L = \sum_{a=1}^{M} \pi_a L_{ar} \ ,$$

where π_a is the frequency of the ath state, and r is the root of the tree. The root can actually be placed on any internal node of the tree because the latter is unrooted [74]. The likelihood of the full data set is:

$$L = \prod_{i=1}^{N} \sum_{a=1}^{M} \pi_a L_{air} \ ,$$

where N is the number of characters. Taking the logarithm of this expression leads to:

$$\ln L = \sum_{i=1}^{N} \ln \left(\sum_{a=1}^{M} \pi_a L_{air} \right) \ . \tag{5.10}$$

A further layer of complexity is added by considering heterogeneity among sites. Two types of heterogeneity are commonly considered: partitions and mixtures [247, 316]. With partitions, the different sites are assigned in different categories, whereas with mixtures we assume that there are different categories, but we do not know which sites belong to which categories. Denote as f_k the frequency of the kth category in the mixture (with $\sum_k f_k = 1$), then (5.9) would become:

$$L_{aiz} = \sum_{k} f_k \left(\sum_{b=1}^{M} p_{ab}^{k}(t_{xz}) L_{bix} \right) \left(\sum_{b=1}^{M} p_{ab}^{k}(t_{yz}) L_{biy} \right) \ . \tag{5.11}$$

The exponent k of p indicates that these probabilities depend on the categories of the mixture.

The presence of partitions is ignored in this formulation, but they can be taken into account easily because the log-likelihood is summed over all sites: the full log-likelihood would become a sum of individual log-likelihoods similar to (5.10) for each partition (see the section on partitioned models below).

phangorn provides a wide array of tools to estimate phylogenies by maximum likelihood. Its main function is `pml`:

```
pml(tree, data, bf = NULL, Q = NULL, inv = 0, k = 1,
    shape = 1, rate = 1, model = "", ...)
```

where `tree` is an object of class `"phylo"` and `data` is an object of class `"phyDat"`. Recall that this data class may store several kinds of discrete data: nucleotides, amino acids, or user-defined. By default the frequencies of the states are taken as equal unless `bf` is used; it is the same for the rate matrix unless `Q` is used. The three next arguments parameterize inter-site variation in substitution rate as commonly used in phylogenetic analyses [315]: `inv` is the proportion of invariant sites, `k` is the number of categories with different substitution rates, and `shape` is the parameter of the Γ distribution. These two forms of inter-site variation are mixtures because we do not know the category each site belongs to. When the data are amino acids, `model` specifies the substitution model. Finally, '`...`' is used to pass some arguments to other functions.

`pml` calculates the likelihood of the data given the tree and the model specified. Coming back to our two-tip, two-nucleotide data above, we can compute its likelihood as we did by hand:

```
> tr <- read.tree(text = "(y:2,x:1);")
> x <- matrix(c("a", "a", "a", "t"), 2, 2)
> rownames(x) <- c("x", "y")
> x <- as.phyDat(x)
> pml(tr, x, rate = .05/.25)

 loglikelihood: -5.288237

unconstrained loglikelihood: -1.386294

Rate matrix:
  a c g t
a 0 1 1 1
c 1 0 1 1
g 1 1 0 1
t 1 1 1 0

Base frequencies:
```

```
0.25 0.25 0.25 0.25
```

If we use (5.10) above we find -5.368 which is quite close but not equal. This is because `pml` uses standard formulae for all models and parameterizations. The rate parameters in Q are multiplied by the base frequencies before computing the probability matrix: this is why we used `rate = .05/.25` (i.e., α/π).

The "unconstrained loglikelihood" is a measure of the maximally achievable value of the log-likelihood—though it is often smaller than the log-likelihood of the data as shown below—and is independent of the tree.

`pml` does not estimate the parameters in the sense that it does not find their values that maximize the likelihood function. This is done with `optim.pml`:

```
optim.pml(object, optNni=FALSE, optBf=FALSE, optQ=FALSE,
   optInv=FALSE, optGamma=FALSE, optEdge=TRUE, optRate=FALSE,
   control=pml.control(eps=1e-08, maxit=10, trace=1),
   model=NULL, subs=NULL, ...)
```

`object` is an output from `pml`, and the arguments of the form `opt*` specify which parameters to estimate. Very logically, we cannot estimate the parameters in our two-tip example, so we take the woodmouse data and its NJ tree:

```
> tw <- nj(dist.dna(woodmouse))
> x <- as.phyDat(woodmouse)
> o1 <- pml(tw, x)
Warning message:
In log(siteLik) : NaNs produced
> o2 <- optim.pml(o1)
optimize edge weights:  -1867.307 --> -1857.168
optimize edge weights:  -1857.168 --> -1857.168
optimize edge weights:  -1857.168 --> -1857.168
> o2

 loglikelihood: -1857.168

unconstrained loglikelihood: -1866.991

Rate matrix:
  a c g t
a 0 1 1 1
c 1 0 1 1
g 1 1 0 1
t 1 1 1 0

Base frequencies:
0.25 0.25 0.25 0.25
```

(The warning message issued after the first call of `pml` is caused by the fact that `tw` has one branch length negative; this problem is solved once we estimated them by maximum likelihood.) We used the default options of `optim.pml`, so only the branch lengths have been estimated. Thus we effectively used the JC69 model. The log-likelihood has been increased from −1867 to −1857. *We have changed the parameter values in order to maximize the likelihood.* This is the normal estimation procedure and there are no reason to compare the two values here. Now let us try to estimate the substitution rate together with the branch lengths:

```
> ctr <- pml.control(trace = 0)
> optim.pml(o1, optRate = TRUE, control = ctr)

 loglikelihood: -1857.168
....

Warning message:
you can't optimise edges and rates at the same time,
 only edges are optimised
```

The warning issued by `phangorn` is clear: the substitution rate α cannot be identified separately from the branch lengths. Of course, we could estimate the rate if we assume the branch lengths to be known:

```
> o3 <- optim.pml(o1, optRate=TRUE, optEdge=FALSE, control=ctr)
> o3$rate
[1] 1.105272
```

Thus using the branch lengths as estimated by NJ, the maximum likelihood estimate is $\hat{\alpha} = 1.105$. However, the branch lengths estimated by NJ have also time and substitution rate confounded; such estimates of rates are more useful when branch lengths are in absolute time units.

`optim.pml` allows us to define the model by specifying which parameters are to be estimated. To see this, let us fit the GTR model:

```
> o4 <- optim.pml(o1, optBf=TRUE, optQ=TRUE, control=ctr)
> o4

 loglikelihood: -1756.274

unconstrained loglikelihood: -1866.991

Rate matrix:
             a         c         g            t
a 0.000000e+00  1.455591 22.192215 5.133905e-05
c 1.455591e+00  0.000000  2.332634 1.964957e+01
g 2.219222e+01  2.332634  0.000000 1.000000e+00
```

Table 5.4. Nucleotide substition models in phangorn in order of decreasing complexity. The second column shows how rates of the GTR model are reduced to simplify the models

Model	Substitution rates	Base frequencies
GTR	$\alpha, \beta, \gamma, \delta, \epsilon, \zeta$	Unbalanced
SYM	"	Balanced
TVM	$\alpha = \zeta$	Unbalanced
TVMe	"	Balanced
TIM3	$\beta = \delta, \gamma = \epsilon$	Unbalanced
TIM3e	"	Balanced
TIM2	$\beta = \gamma, \delta = \epsilon$	Unbalanced
TIM2e	"	Balanced
TIM1	$\beta = \epsilon, \gamma = \delta$	Unbalanced
TIM1e	"	Balanced
TPM3u	$\alpha = \zeta, \beta = \delta, \gamma = \epsilon$	Unbalanced
TPM3	"	Balanced
TPM2u	$\alpha = \zeta, \beta = \gamma, \delta = \epsilon$	Unbalanced
TPM2	"	Balanced
TPM1u	$\alpha = \zeta, \beta = \epsilon, \gamma = \delta$	Unbalanced
K81 (TPM1)	"	Balanced
TrN	$\beta = \gamma = \delta = \epsilon$	Unbalanced
TrNe	"	Balanced
HKY	$\alpha = \zeta$, $\beta = \gamma = \delta = \epsilon$	Unbalanced
K80	"	Balanced
F81	$\alpha = \beta = \gamma = \delta = \epsilon = \zeta$	Unbalanced
JC	"	Balanced

```
t 5.133905e-05 19.649572  1.000000 0.000000e+00

Base frequencies:
0.3042873 0.2662815 0.1288796 0.3005517
```

What we have done now is very different from above: *We have added parameters in the model: this will necessarily increase the likelihood.* This increase is because we gave more freedom to the model to fit the data by adding parameters. A statistical test may tell us whether this increase is significant or not. Instead of rushing into the tests, let us have a look at the rate matrix. The diagonal is set to 0, but this is arbitrary. Like for the JC69 model, rates cannot be identified separately from the edge lenths, however, if there are several rates their relative values may be estimated. So here the rate of the substituion G $\leftrightarrow$ T is fixed and, e.g., the rate of A $\leftrightarrow$ G is estimated to be 22 times higher.

It is critical to examine such outputs before doing the tests, because they give more than useful information on the processes. Now, we may wonder if the increase in log-likelihood from -1857 to -1756 is statistically significant.

We know that twice this difference in log-likelihood follows a χ^2 with degrees of freedom given by the number of additional parameters (this is a likelihood ratio test, LRT). We have added eight parameters (five rates and three base frequencies), so the LRT is $\chi^2_8 = 202$. The P-value may be calculated with `1 - pchisq(202, 8)`. Getting these differences correctly may be sometimes tedious, and the generic function `anova` is helpful here:

```
> anova(o2, o4)
Likelihood Ratio Test Table
  Log lik. Df Df change Diff log lik. Pr(>|Chi|)
1  -1857.2 27
2  -1756.3 35         8        100.89  < 2.2e-16
```

It is left to the user to insure that the correct models are passed to `anova`. For instance, it would not be meaningful to do `anova(o1, o4)` because the branch lengths were not estimated by maximum likelihood in `o1`.

Now we incorporate inter-site variation with the usual Γ distribution: we have to redefine a new model with `pml` and estimate α:

```
> o5 <- optim.pml(pml(tw, x, k = 4), optBf=TRUE, optQ=TRUE,
+                 optGamma = TRUE, control = ctr)
> o5

 loglikelihood: -1743.367

unconstrained loglikelihood: -1866.991
Discrete gamma model
Number of rate categories: 4
Shape parameter: 0.004467235

Rate matrix:
             a         c         g            t
a 0.000000e+00  1.476878 23.294362 6.595230e-05
c 1.476878e+00  0.000000  2.566102 2.010049e+01
g 2.329436e+01  2.566102  0.000000 1.000000e+00
t 6.595230e-05 20.100486  1.000000 0.000000e+00

Base frequencies:
0.3054005 0.2646932 0.1279518 0.3019545
```

The fact that substitution rates vary substantially within coding sequences has been widely documented in the literature, so we anticipate the estimate of the shape parameter $\hat{\alpha}_\Gamma = 0.0045$ (the subscript Γ is to avoid confusion with the substitution rate α) to be statistically significant. Note that if $\alpha_\Gamma \to \infty$, then there is no intersite variation in substitution rate.

```
> anova(o4, o5)
```

```
Likelihood Ratio Test Table
  Log lik. Df Df change Diff log lik. Pr(>|Chi|)
1  -1756.3 35
2  -1743.4 38         3        25.820  1.040e-05
```

There are many other models that we may fit to these data. The choice of the fitted models must be driven by biological questions—as far as the evolutionary biologist is concerned. Figure 5.14 and Table 5.5 give general guidelines on how to conduct phylogeny estimation.

Partitioned Models

`pmlPart` is the function in `phangorn` to fit partitioned models. It is used with `pmlPart(`*`global ~ local`*`, object, ...)` with *`global`* being the components of the model that are common across partitions, *`local`* are the components specific to each partition, and `object` is an object of class `"pml"`. For instance, `pmlPart(edge ~ shape, obj)` will estimate branch lengths common to all partitions and an α_Γ specific to each one. Only the parameters whose components are given in this formula are estimated by `pmlPart`. Before calling the function, we have to specify whose partition each site belongs to. The procedure is slightly tedious, so we detail it here. For the woodmouse data it makes sense to define three partitions with the three codon positions. We recall that the 965 sites define 65 patterns across the 15 individuals:

```
> x
15 sequences with 965 character and 65 different site patterns
The states are a c g t
```

It may be useful to remind that the site patterns correspond to the unique columns of the original alignment, and two sites with the same pattern have exactly the same contribution to the likelihood function, so we have to compute (5.10) for $N = 65$ sites instead of 965. The attribute `"index"` keeps track of the pattern each site belongs to:

```
> str(attr(x, "index"))
 num [1:965] 1 2 2 3 4 5 5 5 5 5 ...
```

So the second and third columns in `woodmouse` have the same pattern (i.e., they are identical) which is the second one in `x`, while the first, fourth, and fifth columns define different patterns, etc. We have now to make a contingency table dispatching the 65 site patterns in the three partitions defined by the three codon positions. This can be done with `xtabs` as illustrated in the help page `?pmlPart`, or more directly with `table`:

```
> w <- table(attr(x, "index"), rep(1:3, length.out = 965))
> dim(w)
[1] 65  3
```

```
> w[11:15, ]

      1   2   3
  11  0   0   1
  12  0   0   1
  13 82  60 120
  14 76 125  59
  15 70  73  82
```

This shows that, for instance, the eleventh and twelveth patterns were observed only once on a third position, whereas the thirteenth pattern was observed 82, 60, and 120 times on each position, and so on for the 65 rows of `w`. This table is used as weights in the call to `pmlPart`:

```
> o6 <- pmlPart(~ rate, o2, weight = w, control = ctr)
> o6

loglikelihood: -1827.652

loglikelihood of partitions:
  -530.835 -490.0756 -806.7415
AIC:  3713.304  BIC:  3854.596

Proportion of invariant sites: 0 0 0

Rates:
0.4397931 0.2189536 2.345432
....
```

The improvement in log-likelihood is nearly 30 units for two additional parameters, so this is surely very significant:

```
> pchisq(2*(1857.168 - 1827.652), 2, lower.tail = FALSE)
[1] 1.518323e-13
```

The estimated rates agree with the expectation that the second codon positions are those evolving at the slowest rate while the third ones are the fastest ones.

We have shown in the previous section that the GTR + Γ model fits better to these data than the simple JC69, so we now try this model with different rates and shape parameters (denoted α_Γ) in the three partitions:

```
> o7 <- pmlPart(~ rate + shape, o5, weight = w, control = ctr)
> o7

loglikelihood: -1714.530
```

```
loglikelihood of partitions:
  -529.0336 -473.0926 -712.4034
AIC:  3521.059  BIC:  3745.177

Proportion of invariant sites: 0 0 0

Rates:
0.3793673 0.1884981 2.436596

Shape parameter:
0.005775529 99.99474 0.2169682
....
```

The decrease in log-likelihood is again close to 30, but this time we added four parameters:

```
> pchisq(1743.367 - 1714.530, 4, lower.tail = FALSE)
[1] 8.436594e-06
```

This is again very significant, and suggests that substitution rate is more heterogeneous in the first codon position than in the third one (recall that the smaller the value of α_Γ, the more heterogeneous the substitution rate among sites). However, a more parsimonious model shows that this is not significant with the present data:

```
> o8 <- pmlPart(~ rate, o5, weight = w, control = ctr)
> o8

loglikelihood: -1715.329

loglikelihood of partitions:
  -529.0337 -473.2011 -713.0942
AIC:  3510.658  BIC:  3705.543

Proportion of invariant sites: 0 0 0

Rates:
0.3910165 0.1919223 2.421476
....
```

The increase in log-likelihood of `o7` compared to `o8` is less than one while, to be significant with two degrees of freedom, it needs to be at least three.

Mixture Models

We have seen that including intersite variation in substitution rate with a Γ distribution is a form of mixture. `pmlMix` is a function to model higher order

mixtures. Its syntax is close to the one of `pmlPart`, the differences are the option `m` specifying the number of categories (2 by default), and `omega` is a vector giving their frequencies (f_k in eq. 5.11); by default they are equal and used as starting values for the estimation procedure. Let us take the fitted object `o2` which is a simple JC69 model. We now include a mixture on its rate with four categories:

```
> o9 <- pmlMix(~ rate, o2, m = 4, control = ctr)
> o9

loglikelihood: -1845.219

unconstrained loglikelihood: -1866.991
AIC:  3750.438  BIC:  3815.670


posterior: 0.7313842 0.004928028 0.2476871 0.01600061

Rates:
3.523458e-07 0.3680481 4.030027 1.448669e-05
....
```

The substitution rate varies considerably along the sequence (remind this is cytochrome *b*) and the most frequent category is the one with the lowest rate.

Logically we expect similar results with a traditional Γ distribution. We check it by printing the log-likelihood:

```
> logLik(optim.pml(update(o2, k = 4), optGamma = TRUE,
+   control = ctr))
'log Lik.' -1845.001 (df=30)
```

The values are close but not equal because `o6` did not use a Γ distribution.

It is interesting to compare this result with the one from the partitioned model (`o8`). The likelihoods cannot be compared directly with a LRT because the models have different structures and no relationship of nestedness. However, the AIC values can be compared and show that the partitioned model fits much better. This shows that if the structure in the data can be identified and taken into account explicitly, this leads to much more powerful analyses.

5.2.3 Finding the Maximum Likelihood Tree

The previous section details how to find the parameter values, including branch lengths, that maximize the likelihood function for a given tree topology. Finding the topology that maximizes the likelihood function is a much more difficult problem. Many solutions have been proposed in the literature and not a single one seems superior, though some are more useful than others.

phangorn implements an approach similar to the one developed by Guindon and Gascuel [115] whose principle is to consider only NNIs (a tree rearrangement seen above about FastME) that improves the likelihood. Guindon and Gascuel's idea is to avoid recalculating the likelihood of the whole tree by realizing that the likelihood values at a node, as detailed above, may be computed in two directions in an unrooted tree. After performing an NNI operation, some node likelihoods in one direction will be unchanged, and others in the other direction as well: this limits considerably the calculations to obtain the complete likelihood of the new topology. Besides, there is only one possible NNI for each internal branch, so scanning the tree for all potential improvements is straightforward. These cycles of NNI moves are alternated with cycles of parameter optimizations. The whole procedure is iterated until the tree likelihood is stable. The option `optNni = TRUE` in `optim.pml` will perform this likelihood topology estimation.

phangorn has a significant improvement over PhyML in that partitions can be modeled with `pmlPart` and it is possible to estimate a different topology for each partition with `pmlPart(~ nni, ....)`, meaning that NNI search will be performed independently in each partition. This is what is often called "mixture of trees" in the literature.

Because phylogeny estimation by maximum likelihood requires intensive computations, computer programs to perform this task are the results of a lot of work. phangorn represents a significant achievement with this respect, but it makes sense to write simple interfaces between R and other applications. There already exist several of them in various packages and two of them are relevant here: `phymltest` in ape (detailed in the next section) and `raxml` in phyloch, an interface to RAxML [287].[4]

RAxML was particularly designed for the analysis of large data sets (> 1000 sequences) though it can analyse small data sets as well. It uses a tree rearrangement named lazy subtree rearrangement (LSR) where a subtree is pruned and grafted on a neighboring branch: among all the new topologies explored, only the 20 best trees are fully optimized [288]. This procedure is repeated until no better topology is found. RAxML uses an initial parsimony tree. Another particularity of this program is the use of a GTR–CAT model in place of the usual GTR + Γ: the likelihood are first calculated for each individual site which is then attributed to a category with respect to its individual likelihood contribution. This avoids recalculating the contribution of each site to each category of the Γ distribution. However, the likelihood values calculated with the GTR–CAT model are not comparable with those from traditional models, so RAxML allows one to use the GTR + Γ model on the final topology in order to estimate α_Γ.

The interface from R is:

```
raxml(seq, runs = 10, partition = NULL, outgroup = NULL,
      backbone = NULL, path = "/Applications/RAxML-7.0.4",
```

[4] http://icwww.epfl.ch/~stamatak/index-Dateien/Page443.htm

```
        optimize = FALSE, clear = TRUE, file = "fromR")
```

The data `seq` must be of class `"DNAbin"`. The option `partition` allows to define a partitioned model. The call of `raxml` creates a subdirectory in the directory defined by the argument `path` where the outputs of RAxML are stored and returns a tree of class `"phylo"`:

```
> MLtree <- raxml(woodmouse, path = "~/RAxML-7.0.4/")
....
Overall Time for 10 Inferences 6.447887
Average Time per Inference 0.644789
Average Likelihood   : -1743.700212

Best Likelihood in run number 8: likelihood -1743.528033
....
```

The returned log-likelihood (and not likelihood as printed by RAxML) is very close to the one obtained with `optim.pml` (see `o5`, p. 153).

5.2.4 DNA Mining with PhyML and `modelTest`

The previous section explains how to define and fit a variety of molecular evolution models. How to select the appropriate model(s) for parameter estimation is an issue that has attracted a lot of attention and debate among statisticians [34, 35, 48, 204]. The importance of model selection in a likelihood framework has been made repeatedly in the phylogenetic literature [206, 246, 244]. Posada and Crandall [245] developed a computer program, to be used with the program PAUP*, that fits a series of DNA evolution models to a given data set. This program is supposed to help in selecting a substitution model for further analyses.

In order to provide a similar functionality, but with a free phylogeny estimation program, `ape` has the function `phymltest` which uses PhyML developed by Guindon and Gascuel [115, 139].[5] Another difference is that `phymltest` lets PhyML search for the best tree using NNI moves for all fitted models. All substitution models available in PhyML are used; these are: JC69, K80, F81, F84, HKY85, TN93, and GTR. Additionally, models with(out) invariant sites and / or intersite variation (with the usual Γ distribution) are used. This results in 28 fitted models. The interface is:

```
phymltest(seqfile, format = "interleaved", itree = NULL,
          exclude = NULL, execname = NULL, append = TRUE)
```

where `seqfile` is the name of the file with the sequences (given as a character). The other arguments have default values: `itree` is the initial tree (by default, a BIONJ tree made by PhyML), and `execname` is the name of the PhyML

[5] http://www.atgc-montpellier.fr/phyml/

executable which depends on the operating system (phyml_3.0.1_linux32, phyml_3.0.1_macintel, or phyml_3.0.1_win32.exe). Under Unix-style systems, it is convenient to create a symbolic link named 'phyml', for instance in your home directory, which can be subsequently modified when a new version of PhyML is issued; thus it is not needed to change the R scripts which have `execname = "~/phyml"`.

Some care must be taken to set correctly the three diffferent paths involved here: the path to PhyML's executable, the path to the sequence file, and the path to R's working directory. Here are two possible uses under Linux and Windows:

```
phymltest("/home/paradis/data/seq.txt",
          execname = "phyml", path2exec = "/usr/bin")
phymltest("D:/data/seq.txt", path2exec = "D:/phyml")
```

If R returns an error message because of a problem in finding one of the files, it might be better to move all files in the same directory, say '/home/paradis/phyml' or 'D:/phyml', and set the latter as R's working directory:

```
# Linux:
setwd("/home/paradis/phyml")
phymltest("seq.txt", execname = "phyml_linux")
# Windows:
setwd("D:/phyml")
phymltest("seq.txt")
```

`phymltest` returns an object of class `"phymltest"` that has three methods: the `print` method prints a table of all fitted models with the number of free parameters, the values of the log-likelihood, and the Akaike Information Criterion (AIC); the `summary` method computes and prints all possible likelihood ratio tests (LRTs) between pairs of nested models; and the `plot` method plots, on a vertical axis, all AIC values with an indication of the corresponding model (see Section 5.8 for an example).

phangorn has a similar function, `modelTest`, that fits a series of models with its own functions. The set of default models is slightly different: JC69, F81, K80, HKY85, SYM, GTR, and includes intersite variation with four categories and invariants. These may be controlled with the options `model`, `G`, `I` (both logical), and `k` for the number of categories if `G = TRUE`. `modelTest` returns a data frame with the values of log-likelihood, AIC and BIC (Bayesian Information Criterion). For a simple example with the woodmouse data:

```
> modelTest(x, G = FALSE, I = FALSE)
  Model df    logLik      AIC      BIC
1    JC 27 -1857.165 3768.330 3899.878
2   F81 30 -1810.958 3681.916 3828.080
3   K80 28 -1806.798 3669.596 3806.016
```

```
4   HKY 31 -1759.459 3580.918 3731.954
5   SYM 32 -1803.493 3670.986 3826.894
6   GTR 35 -1756.274 3582.548 3753.073
```

The AIC and BIC values agree that the HKY85 model is the best one for these data, closely followed by GTR, while the four other models do not fit appropriately.

5.3 Bayesian Methods

The methods we have used so far consider that data are produced by a random process whose unknown parameters are constant; the estimates of these parameters are random variables. They are called frequentist methods. Bayesian methods do not consider parameters as constants but as probabilistic distributions. This requires to define these distributions before looking at the data: the priors.[6] Bayesian methods calculate the posterior distributions of the parameters by combining the priors with the likelihood of the data using conditional probabilities with Bayes's theorem (or Bayes rule). Such a calculation involves integrating over the priors of all parameters in the model: this integral has thus as many dimensions as there are parameters. Calculating an integral in a single dimension may be difficult, so calculating an integral in many dimensions is an arduous task. A large number of computational methods have been invented to do these calculations, the most well-known being the Markov chains Monte Carlo (MCMC).

It would take many pages to describe Bayesian phylogenetic methods and the controversy about their use (which is little compared to the same debate among statisticians), but the above paragraph summarizes the fundamental points to keep in mind when using a Bayesian method in phylogenetics, and actually in any field. For a gentle introduction to Bayesian statistics see Albert's *Bayesian Computation With R* [7].

Consider a very simple Bayesian estimation of a phylogeny. We assume a JC69 model, so there is only one substitution parameter (α). Priors must therefore be specified for the tree topology, the branch lengths, and α. We choose the followings: all topologies are equiprobable, branch lengths from a uniform distribution on [0, 0.1], and α from a uniform distribution on [0, 1]. Integrating over all the parameters is not possible with standard mathematics. A solution is to sample this multidimensional space randomly in order to have a general image of its density. The sampling procedure is repeated a large number of times (say 10^6) storing each value α and the log-likelihood (the data `x` is of class `"phyDat"`, and the number of sequences is `n`). We focus here on the estimation of α:

[6] Bayesian priors may be interpreted very differently depending on the meaning attached to them.

```
alpha <- lnL <- numeric(1e6)
for (i in 1:1e6) {
    tr <- rtree(n, rooted = FALSE, tip.label = names(x),
                br = runif, min = 0, max = 0.1)
    alpha[i] <- runif(1)
    lnL[i] <- pml(tr, x, rate = alpha[i])$logLik
}
```

We compute the posterior distribution of α for the woodmouse data with a histogram; we take care to set the x-axis on the original scale of the prior (Fig. 5.5):

```
o <- hist(alpha*lnL/sum(alpha*lnL), 50, xaxt = "n",
          main = "", xlab = expression(alpha))
axis(1, at = seq(min(o$breaks), max(o$breaks), length.out=5),
     labels = seq(min(alpha), max(alpha), length.out = 5))
```

The posterior is highest at $\alpha \approx 0.05$. It can be checked that this distribution is heavily influenced by the priors by modifying the above code.

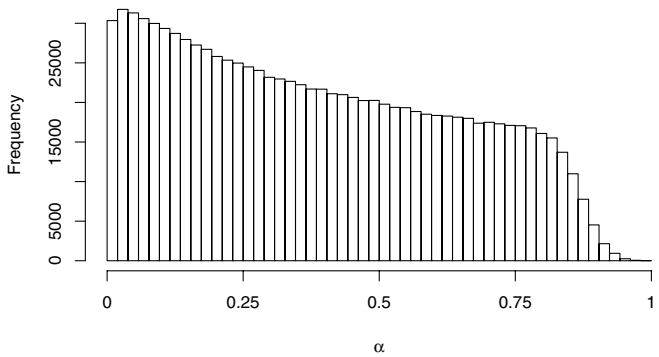

Fig. 5.5. Posterior distribution of α from 10^6 Monte Carlo generations

What we have done is a simple Monte Carlo integration assuming that the algorithm sufficiently explored the parameter space to give a picture of the posterior distribution. In most cases Monte Carlo integration is not efficient because the space is very large and its random exploration spends a lot of time in parts of low likelihood value. A significant improvement is due to Hastings [130] who proposed to avoid exploring the low-likelihood portions of the parameter space with the following rule:

- If the likelihood of the new proposal is higher than the previous one, then accept it.
- If the likelihood of the new proposal is lower, then accept it with probability equal to the ratio of the two likelihoods. If it is rejected, sample the priors for another proposal.

If we try to implement a Metropolis–Hastings version of our algorithm above, the chain would be quickly stuck because the complete random sampling procedure is highly likely to generate data far from the current tree and so will be rejected. To avoid this we sample trees in a correlated way: topologies are modified with a NNI move using the function `rNNI` from phangorn, and branch lengths are modified by adding to each of them a small noise. Still we sample from the same priors by constraining the branch lengths to be between 0 and 0.1. For the sake of simplicity, we continue to sample values of α in a non-correlated way. The code is longer this time because we must initialize the chain with a random tree, and we implement the Hastings ratio. As before, we focus on the estimation of α.

```
alpha <- lnL <- numeric(1e3)
tr <- rtree(n, rooted = FALSE, tip.label = names(x),
            br = runif, min = 0, max = 0.1)
alpha[1] <- runif(1)
lnL[1] <- pml(tr, x, rate = alpha[1])$logLik
i <- 2
while (i <= 1e3) {
    tr2 <- rNNI(tr)
    tr2$edge.length <- tr2$edge.length
          + rnorm(2*n - 3, sd = 0.1)
    tr2$edge.length[tr2$edge.length < 0] <- 0
    tr2$edge.length[tr2$edge.length > 0.1] <- 0.1
    alpha[i] <- runif(1)
    lnL[i] <- pml(tr2, x, rate = alpha[i])$logLik
    accept <- if (lnL[i] >= lnL[i - 1]) TRUE
    else as.logical(rbinom(1, size = 1,
         prob = exp(lnL[i] - lnL[i - 1])))
    if (accept) {
        tr <- tr2
        i <- i + 1
    }
}
```

We run this until we have 1000 proposal trees accepted. Figure 5.6 shows the evolution of the log-likelihood along the chain:

```
plot(lnL, type="l", xlab="Generation", ylab="Log-likelihood")
```

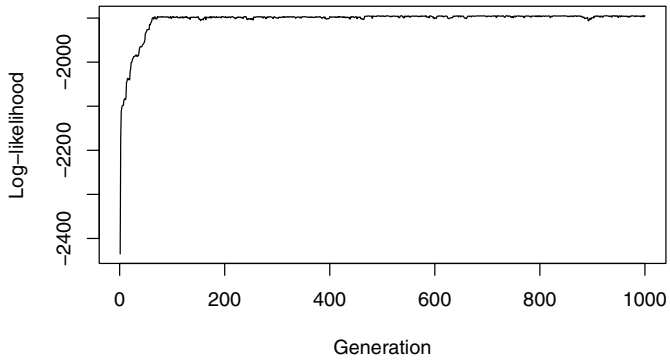

Fig. 5.6. Change in log-likelihood along an MCMC

The same plot for the simple Metropolis algorithm, not shown here, would have displayed wide variation in the log-likelihood of the sampled trees. Figure 5.7 displays the estimated posteriors for α with all samples (left) and after removing the first one hundred ones (right); the code is the same than above. This time the estimates are much more narrower and concentrated around $\alpha \approx 0.066$. In publications, these distribution estimates are plotted as a smoothed curve (see `?density` on how to smooth easily a histogram, or the package KernSmooth for more sophisticated methods).

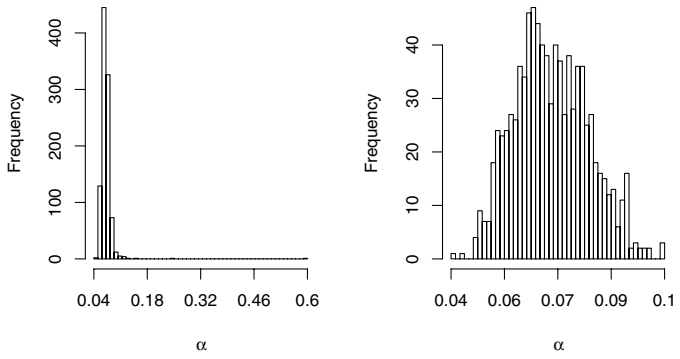

Fig. 5.7. Posterior distribution of α with all samples (left) and after removing the first one hundred ones (right)

This very simple example gives a brief view of the complexity of Bayesian estimation in a situation with many parameters which is the case in phylogenetics. Checking the impact of the priors, the convergence of the chains, the impact of the sampler used should all be done routinely, but this is a lot work. Often (too often in my opinion), a Bayesian analysis is run more to contemplate the final tree rather than to give insights into the biological processes that shaped the data. I recommend to spend a comparable effort into graphical exploratory analyses, examination of the alignment(s) and of the distribution of pairwise distances, quantification of inter-site variation in substitution rate, and assessment of support and conflict values from bootstrap trees. This list is not limitative and Fig. 5.14 gives an overview of this process.

The function `mrbayes` in phyloch provides an interface to MrBayes [142]:

```
mrbayes(x, file, nst = 6, rates = "invgamma", ngammacat = 4,
  nruns = 2, ngen = 1e+06, printfreq = 100, samplefreq = 10,
  nchains = 4, savebrlens = "yes", temp = 0.2, burnin = 10,
  contype = "allcompat", path="/Applications/mrbayes-3.1.2/",
  run = TRUE)
```

`mrbayes.mixed` has exactly the same options: it considers morphological data.

5.4 Other Methods

5.4.1 Parsimony

The parsimony principle is extremely simple: assume the least number of evolutionary changes through time.[7] For instance, let us go back to the data on Fig. 5.4 and take the first site: it is the same for x and y. So the parsimony principle leads us to say that z had A as well on this site. This reasoning involves no parameter and no branch lengths. We have seen in Section 5.2.2

[7] The principle of parsimony as it is applied to phylogenetics must be distinguished from the principle of the same name in statistics. The latter says that non-significant effects should not be included in a model. For instance, the linear model $y = \beta_1 x_1$ is preferred to $y = \beta_1 x_1 + \beta_2 x_2$ if we find that β_2 is not significantly different from zero. There are two fundamental differences between these two principles of parsimony. First, statistical parsimony is data-driven: the choice of the model depends on the data at hand. Phylogenetic parsimony is an a priori concept which does not require any data. Second, statistical parsimony is more closely related to the principle of not invoking unnecessary causes known as Occam's razor. A single cause, high mutation rate, may result in many changes in a molecular sequence. Parsimony in process does not imply parsimony in pattern. On the other hand, parsimony analysis of complex characters (eye, lung, ...) is meaningful because such characters evolve slowly and, certainly, each with different causes, but this logic applies poorly to DNA sequences.

that the likelihood approach does not ignore the fact that z may not have A (z is not observed) but this requires to assume an evolutionary model.

Let us consider the second site now (recall that it is A in x and T in y). The likelihood calculations followed logically, but what about applying parsimony in this case? Clearly, we have to assume that one change occurred. But on which branch? Because branch lengths do not exist in parsimony, this is a problem.

Suppose that we observe the same second site on many other species (x and y are still sister-species). If A is observed in these species, then we will conclude that a change occurred on the branch $z \rightarrow y$. If T is observed instead, then we will conclude that a change occurred on the branch $z \rightarrow x$. The likelihood calculations for z will remain the same in both cases.

This illustrates a property of parsimony. When polymorphism is low, parsimony calculations are relatively easy, but when this is not the case, they are problematic.

phangorn has the function `parsimony` that computes the parsimony score of a tree (or a list of trees) and some data of class `"phyDat"`. Two methods are available through the option `method`: `"sankoff"` (the default) and `"fitch"`.

```
> parsimony(tw, x)
[1] 68
```

`optim.parsimony` searches for the maximum parsimony tree by performing NNIs on the topology; the branch lengths are in number of inferred changes:

```
> pars.tr <- optim.parsimony(tr, x)
optimize topology:  68 --> 68
Final p-score 68 after  0 nni operations
> pars.tr$edge.length
 [1] 1 1 1 0 1 3 1 6 9 5 3 2 2 3 3 0 4 5 1 2 5 2 1 1 1 2 3
```

Thus the MP tree has the same topology than the NJ tree. We remember that the ML tree inferred on page 153 has a slightly different topology than the NJ one:

```
> parsimony(o5$tree, x)
[1] 68
```

So the ML tree has the same parsimony score.

The functions `CI` and `RI` calculate the consistency and retention indices:

```
> CI(tw, x)
[1] 3.191176
> RI(tw, x)
[1] -0.4056604
```

5.4.2 Hadamard Conjugation

Methods of phylogenetic inference based on the Hadamard conjugation have the advantage of permitting direct calculations of the quantities of interest, instead of the recursive computations needed for likelihood or distance-based methods. Their weaknesses however prevent their general use. Only simple substitution models can be handled: the most complex model that can be analyzed is K81. Furthermore, though the calculations are direct they cannot handle a large number of observations (see below). This section will outline the principle of these methods and illustrate the functions available in phangorn.

The principle of Hadamard methods is to calculate the frequency distribution of the site patterns among the observations. To illustrate this, we take four sequences from the woodmouse data that we transform in two-state characters (purine and pyrimidine, R and Y) with the function `acgt2ry`:

```
> X <- acgt2ry(as.phyDat(woodmouse[12:15, ]))
> X
4 sequences with 915 character and 5 different site patterns
The states are r y
```

To see what are the five site patterns, we print the structure of these data:

```
> str(X)
List of 4
 $ No1114S: int [1:5] 1 2 1 2 2
 $ No1202S: int [1:5] 1 2 1 1 2
 $ No1206S: int [1:5] 1 2 1 1 1
 $ No1208S: int [1:5] 1 2 2 1 2
....
 - attr(*, "weight")= int [1:5] 393 518 1 2 1
....
```

The first observed pattern is R in all four sequences and it was observed 393 times; the second pattern is Y everywhere and it was observed 518 times, and so on.

Hendy and Penny [133] developed a series of equations that allow calculation of the distribution of the site patterns for a given tree assuming a model of equal rate of change R $\leftrightarrow$ Y. By reversing these equations and using the observed site pattern frequencies, it is possible to estimate the branch lengths of the tree. Different trees can be compared with a least squares criterion.

The function `h2st` in phangorn computes the Hadamard conjugation under this model. It is limited to analyze no more than 23 sequences. The calculations are based on identifying the site patterns that define a split: the sequences with R all on one side, and those with Y on the other. With two states, there are 2^{n-1} possible splits, here $n = 4$, so $2^{4-1} = 8$ (taken from the internal code of `h2st`):

```
> sv
[1] 911   2   0   0   1   0   0   1
```

These are the numbers of sites defining each split. So there are 911 sites definig the first split (393 + 518, as seen in `"weight"`), and so on. For each of these splits, one internal branch is defined, and its length is computed with the fast Walsh–Hadamard transform:

```
> cbind(sv, qv)
      sv            qv
[1,] 911  6.814539e+00
[2,]   2  2.195398e-03
[3,]   0  5.290676e-09
[4,]   0 -1.204954e-06
[5,]   1  1.097702e-03
[6,]   0 -2.409891e-06
[7,]   0 -2.409891e-06
[8,]   1  1.097702e-03
```

Logically, the splits defined by site patterns not observed in the data have very small or negative internal branch lenths. `h2st` drops the first split (with all observations) and those whose branch length is below a value given by its option `eps` (0.001 by default):

```
> h2st(X)
[[1]]
[1] 1

[[2]]
[1] 3

[[3]]
[1] 1 2 3

attr(,"weights")
[1] 0.002195398 0.001097702 0.001097702
attr(,"labels")
[1] "No1114S" "No1202S" "No1206S" "No1208S"
attr(,"class")
[1] "splits"
```

`h4st` implements the Hadamard conjugation with the K81 model. It is limited to $n \leq 11$. The calculations are more complicated and we take only three sequences to shorten the output:

```
> h4st(as.phyDat(woodmouse[13:15, ]))
$q
```

```
     transversion  transition.1  transition.2
[1,]  0.003125601 -1.077571e-06  3.019600e-08
[2,]  0.005269232  1.060610e-03 -5.553509e-06
[3,]  0.010586204  1.038186e-03  2.110004e-03

$qv
            [,1]          [,2]          [,3]          [,4]
[1,] 6.848987958  3.019600e-08 -5.553509e-06  2.110004e-03
[2,] 0.003125601 -1.077571e-06 -1.120762e-05 -1.206513e-05
[3,] 0.005269232 -2.226369e-06  1.060610e-03 -1.436271e-05
[4,] 0.010586204  2.527083e-08 -3.255929e-06  1.038186e-03

$sv
     [,1] [,2] [,3] [,4]
[1,]  943    0    0    2
[2,]    3    0    0    0
[3,]    5    0    1    0
[4,]   10    0    0    1

$n
[1] 965

$names
[1] "No1202S" "No1206S" "No1208S"
```

5.4.3 Species Trees *Versus* Gene Trees

Molecular phylogenies allow us to trace back the history of genetic lineages, but these may not coincide completely with the history of species because, as geneticists have demonstrated some time ago, different genes may have different histories [163, 165]. The problem is particularly acute for recently diverged species: Kimura and Ohta's classical result showed that it takes some time for two populations to diverge by genetic drift, so that some shared molecular polymorphism can be the result of recent inherante from their ancestors [164]. This process is called *incomplete lineage sorting* [193].

A classical text-book image shows connected tubes (the species tree) with lines inside (the gene trees) that diverge before or after the tubes diverged themselves. The distribution of the gene trees depends on the coalescent and has attracted a lot of attention in recent years [e.g., 52, 243]. Several methods developed to estimate a species tree from a set of gene trees share a common framework:

1. For each gene tree, compute the cophenetic distance matrix; this gives for each pair of taxa a series of distances derived from each gene.
2. Summarize this series of distances resulting in a new distance matrix for all taxa.

3. Compute a hierarchical clustering from this matrix using a single-linkage method.

This framework is implemented in the function `speciesTree`:

```
speciesTree(x, FUN = min)
```

`x` is a list of (gene) trees, and `FUN` is the function used in step 2 above. The default calculates the maximum tree of Liu et al. [185]. If `FUN = sum`, then the species tree is calculated with the shallowest divergence tree of Maddison and Knowles [193]. The trees in `x` must be ultrametric and with branch lengths on the same temporal scale. Edwards et al. [65] showed using simulations that the gene-tree approach performs much better than the concatenation of sequences (supermatrix) approach, particularly even if most gene trees are discordant with the species tree. However, the gene-tree approach seems well suited when there are many genes and few species.

We take a simple hypothetical example with three species and two gene trees that have the same topology (though the gene trees may have different topologies):

```
> t1 <- read.tree(text = "(c:2,(a:1,b:1):1);")
> t2 <- read.tree(text = "(c:4,(a:2,b:2):2);")
> tmax <- speciesTree(list(t1, t2))
> all.equal(tmax, t1)
[1] TRUE
```

So here the maximum tree is the shortest of the two gene trees. This seems logical in the sense that oldest gene splits predate the species divergence. The shallowest divergence tree is different (Fig. 5.8):

```
> tsha <- speciesTree(list(t1, t2), sum)
> kronoviz(list(t1, t2, tmax, tsha), type = "c")
```

5.5 Bootstrap Methods and Distances Between Trees

The use of the bootstrap has enjoyed great success in phylogenetic analyses [75]. The idea of the bootstrap can be sketched as follows: suppose we are interested in quantifying the confidence level in a parameter estimate given some data, but the expected distribution of this parameter does not hold (because, e.g., of a small sample size, or non-independence of observations). Then we could resample the data at hand many times, mimicking the process of sampling the real population. The variation in the estimated parameter from the "bootstrap" samples is a measure of the confidence level in this estimate [66].

The idea is simple, intuitive, and elegant, but, in some situations, requires intensive computations [69]. The application of the bootstrap in phylogeny

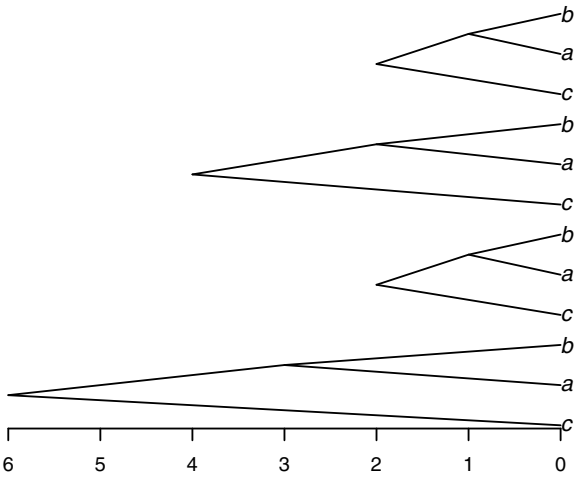

Fig. 5.8. Two gene trees (`t1` and `t2` in the text) on top, the maximum tree, and the shallowest divergence tree on bottom

estimation is almost as simple: estimate a tree with a given method, resample the original data (the matrix taxa × characters) a large number of times, and analyze these "bootstrap" samples with the same method, and calculate the number of times the clades observed in the estimated tree appear in the "bootstrap" ones.

The application of the bootstrap to assess confidence levels in phylogenetic estimation has been criticized, but Efron, Halloran, and Holmes [68] showed that this was due to confusion in the interpretation of the original bootstrap method by Felsenstein [75]. Efron et al. also proposed another way to compute the bootstrap values for hypothesis testing rather than assessing confidence levels [68]. Shimodaira and Hasegawa [283] developed a similar test to compare different topologies with a bootstrap resampling and a likelihood ratio test (see the function `SH.test` in `phangorn`).

In this section, we examine the different ways of resampling phylogenetic data, comparing (possibly a large number of) phylogenetic trees, and computing bootstrap values.

5.5.1 Resampling Phylogenetic Data

R has a powerful function, `sample`, that can be used to create a bootstrap sample from a data set: this function returns a sample, by default without replacement, of the vector given as argument. If the option `replace = TRUE` is used, then sampling is done with replacement which is clearly what is needed

for a bootstrap sample. Below is a simple example with a vector `x` containing 10 values 1, 2, ..., 10:

```
> x <- 1:10
> sample(x)
 [1]  9  8  6  1 10  7  5  4  3  2
> sample(x, replace = TRUE)
 [1]  7  5  2  4 10  6  2  1  2  2
```

Note that `sample(x)` returns a random permutation of the data. We can also give a single integer value to `sample`, say 10, which will then return a sample of integers from 1 to 10.

With phylogenetic data we are mostly interested in resampling the columns of the matrix taxa × characters (where taxa are the rows, and characters the columns). If this matrix is called `X`, then one can simply do:

```
X[, sample(ncol(X), replace = TRUE)]
```

Note the presence of the comma just after the left bracket which means that all rows of `X` will be selected (see p. 17). Here is an example of how this could be used:

```
> x <- scan(what = "")
1: a a c t t a a c t t c a c c t
16:
Read 15 items
> X <- matrix(x, 3, 5, byrow = TRUE)
> X
     [,1] [,2] [,3] [,4] [,5]
[1,] "a"  "a"  "c"  "t"  "t"
[2,] "a"  "a"  "c"  "t"  "t"
[3,] "c"  "a"  "c"  "c"  "t"
> X[, sample(ncol(X), replace = TRUE)]
     [,1] [,2] [,3] [,4] [,5]
[1,] "a"  "c"  "c"  "a"  "a"
[2,] "a"  "c"  "c"  "a"  "a"
[3,] "a"  "c"  "c"  "a"  "c"
```

It happens sometimes that the columns of a matrix are affected with weights, for instance, because the same values have been observed several times for all taxa [68, 168]. This may be a useful way to reduce the size of the data matrix, particularly if few sites are polymorphic. In these cases, resampling must take these weights into account. Suppose each column of `X` is associated with a weight stored in a vector `w` (`length(w)` is equal to `ncol(X)`), then a bootstrap sample is obtained using the option `prob` of `sample`:

```
X[, sample(ncol(X), replace = TRUE, prob = w)]
```

The values passed to `prob` need not sum to 1 because they are used as relative probability weights. If the values in `w` are integer weights, one may need to use the option `size` to produce a sample of the appropriate size:

```
X[, sample(ncol(X), replace = TRUE, prob = w, size = sum(w))]
```

An issue in resampling phylogenetic data is that the columns may not be independent, particularly in the case of molecular sequences. A solution is to sample the sites by groups (or blocks) rather than individually. There are several ways to do this in R. A general solution is to sample the indices of the vector instead of the vector itself. Let us consider the case of sampling blocks of three nucleotides in a vector `x`. First, build a vector with the indices 3, 6, ...:

```
> x <- scan(what = "")
1: a a a c c c g g g t t t
13:
Read 12 items
> x
 [1] "a" "a" "a" "c" "c" "c" "g" "g" "g" "t" "t" "t"
> block <- 3
> i <- seq(block, length(x), block)
> i
[1]  3  6  9 12
```

Then, sample this vector `i` as before:

```
> iboot <- sample(i, replace = TRUE)
> iboot
[1] 12  6 12  9
```

What we want in fact is a vector with the values 10, 11, 12, 4, 5, 6, 10, 11, 12, 7, 8, and 9. The pattern is clear: the 3rd, 6th, 9th, and 12th values are those in `i.boot`, the 2nd, 5th, 8th, and 11th ones can be obtained with `i.boot - 1`, and the 1st, 4th, 7th, and 10th ones can be obtained with `i.boot - 2`. Binding these three vectors in a matrix converted then in a vector is a simple way to proceed:

```
> boot.ind <- as.vector(rbind(iboot - 2, iboot - 1, iboot))
> boot.ind
 [1] 10 11 12  4  5  6 10 11 12  7  8  9
```

The bootstrap sample is finally obtained with:

```
> x[boot.ind]
 [1] "t" "t" "t" "c" "c" "c" "t" "t" "t" "g" "g" "g"
```

Note that we did not use the value of `block` (3) or `length(x)` (12) in the above commands, so they can be used in different situations. They also can be used to resample blocks of columns of a data matrix: in this case it is necessary to replace `length(x)` by `ncol(x)`, and the final command by `x[, boot.ind]`.

Because in most cases, a large number of bootstrap samples will be needed, it is useful to include the appropriate sampling commands in a loop and / or a function. This is what is done by the function `boot.phylo` described below.

For data of class `"phyDat"`, the above scheme cannot be used because the characters are compressed: those with the same pattern among taxa are not repeated. phangorn has two functions to perform a bootstrap on this data class:

```
bootstrap.pml(x, bs = 100, trees = TRUE, ...)
bootstrap.phyDat(x, FUN, bs = 100, ...)
```

The first function does the bootstrap with maximum likelihood estimation, while the second has the argument `FUN` similar to `boot.phylo` (see below). In both cases, a list of `bs` trees is returned.

When doing a bootstrap, it is not always trivial what needs to be resampled: the rows or the columns of a data matrix? What we resample depends on the assumptions of the model. In a phylogeny estimation, the rows of the data matrix (the sequences, the species, or the populations) are not assumed to be independent because of their phylogenetic relationships. The characters (the columns of the matrix) are assumed to evolve independently of each other, so these are the elements to be resampled. In case we do not want to assume independent evolution of the characters, resampling of independent blocks of characters may be done.

5.2.2 Bipartitions and Computing Bootstrap Values

Once bootstrap samples and trees have been obtained, it is necessary to summarize the information from them. ape and phangorn provide several functions for this task depending on the approach taken.

A bipartition is similar to a split: it is made with two subsets of the tips of a tree as defined by an internal branch. `prop.part` takes as its argument a list of trees and returns an object of class `"prop.part"` which is a list of all observed bipartitions together with their frequencies. There are `print` and `summary` methods for this class; the latter prints only the frequencies. Here is the result with a four-taxa tree:

```
> tr <- read.tree(text = "((a,(b,c)),d);")
> prop.part(tr)
==> 1 time(s):[1] a b c d
==> 1 time(s):[1] a b c
==> 1 time(s):[1] b c
```

Instead of a list of bipartitions indexed to the internal branches, `prop.part` returns a list indexed to the numbers of the nodes, and gives the tips that are descendants of the corresponding node: thus the first vector in the list includes all tips because the first node is the root. It is then straightforward to get the bipartitions. The following code prints them for an object named `Y`:

```
for (i in 2:length(Y)) {
    cat("Internal branch", i - 1, "\n")
    print(Y[[i]], quote = FALSE)
    cat("vs.\n")
    print(Y[[1]][!(Y[[1]] %in% Y[[i]])], quote = FALSE)
    cat("\n")
}
```

`prop.clades` takes two arguments: a tree (as a `"phylo"` object), and either a list of trees, or a list of bipartitions as returned by `prop.part`. In the latter case, the list of bipartitions must be named explicitly (e.g., `prop.clades(tr, part = list.part)`). This function returns a numeric vector with, for each clade in the tree given as first argument, the number of times it was observed in the other trees or bipartitions. For instance, we have the obvious following result:

```
> prop.clades(tr, tr)
[1] 1 1 1
```

Like the previous function, the results are indexed according to the node numbers.

Using the two functions just described, bootstrap samples obtained as described in the previous section, and the appropriate function(s) for phylogeny estimation, one can perform the bootstrap for the estimated phylogeny in a straightforward way using basic programming techniques. However, to do such an analysis directly, the function `boot.phylo` can be used instead. Its interface is:

```
boot.phylo(phy, x, FUN, B = 100, block = 1, trees = FALSE,
           quiet = FALSE, rooted = FALSE)
```

with the following arguments:

`phy`: an object of class `"phylo"` which is the estimated tree;
`x`: the original data matrix (taxa as rows and characters as columns);
`FUN`: the function used to estimate `phy` from `x`. Note that if the tree was estimated with a distance method, this must be specified as something such as:

```
FUN = function(xx) nj(dist.dna(xx))
```

or:

```
FUN = function(xx) bionj(dist.dna(xx, "TN93"))
```

`B`: the number of bootstrap replicates;
`block`: the size of the "block" of columns, that is, the number of columns that are sampled together during the bootstrap sampling process (e.g., if `block = 2`, columns 1 and 2 are sampled together, the same for columns 3 and 4, 5 and 6, and so on; see above);
`trees`: a logical value indicating whether to return the `B` bootstrap trees;
`quiet`: a logical value indicating whether to display a progress bar;
`rooted`: a logical value indicating whether to treat the trees as rooted or not. If `FALSE`, the splits are counted; if `TRUE`, the clades are counted.

`boot.phylo` returns exactly the same vector as `prop.clades`. By default, the bootstrap trees are not saved, and so cannot be examined or further analyzed, for instance, to perform the two-level bootstrap procedure developed by Efron et al. [68]. Setting `trees = TRUE` allows to return the bootstrap trees: `boot.phylo` then returns a list with the bootstrap values and the trees.

Why would we want to examine bootstrap trees when we have the bootstrap values right at the nodes of the estimated tree? If these bootstrap values are high, then this is fine, but if they are low this may indicate either a lack of information in the data or a systematic bias in the estimation. A lack of information will be revealed by the fact that bipartition frequencies are more or less random among the bootstrap trees. On the other hand, if a bipartition appears at a high frequency, which would not be visible in the bootstrap values because it is absent in the estimated tree, this may be evidence for a bias, maybe due to a few sites that are resampled rarely during the bootstrap. In this case, using a different approach, for instance with partitions or mixtures, may be useful.

A way to analyze a list of trees and the potential conflict among them is provided by the Lento plot [182]. The function `lento` in `phangorn` analyzes a list of trees provided as an object of class `"splits"`. Two quantities are calculated for each split:

- 'support' which is equal to the observed frequency of the split (as given by the attribute `"weights"` in the class `"splits"`, or the attribute `"number"` in the class `"prop.part"`);
- 'conflict' which is the sum of the support values of the splits that are not compatible with the considered one.

Two splits are not compatible if they cannot be observed in the same binary tree (e.g., AB|CD and A|BCD are compatible, while AB|CD and AC|BD are not). For illustration, we do a bootstrap analysis on the woodmouse data and store the bootstraped trees:

```
> f <- function(x) nj(dist.dna(x))
> tw <- f(woodmouse)
> o <- boot.phylo(tw, woodmouse, f, trees = TRUE)
  |================================================| 100%
```

```
> o
$BP
 [1] 100  24  54  58  61  50  78  65  87  89  95  99  59

$trees
100 phylogenetic trees
```

We extract the splits and their frequencies; we can do this operation with `prop.part` but we prefer `as.splits` so the result can be input directly into `lento`:

```
> s <- as.splits(o$trees)
> length(s)
[1] 90
```

So there are 108 different splits observed among the 100 bootstrap trees. We can now do the Lento plot (Fig. 5.9):

```
> lento(s, trivial = TRUE)
```

The splits are ordered from left to right with decreasing support value indicated as positive. The conflict values are indicated as negative. The structure of each split is displayed with plot symbols. The first sixteen splits are the trivial ones: the fifteen defined by the terminal branches and the one with all tips. They cannot conflict with any other, and so are obviously observed in all trees—they are not plotted by default. It is interesting to note in the present example that, apart the trivial splits, only one has a non-zero conflict value (No1208S, No1007S, No0909S), which may result from the low divergence of these sequences.

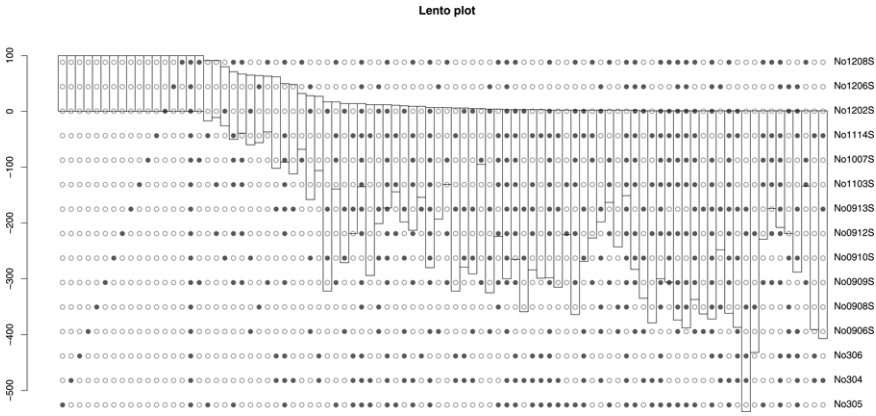

Fig. 5.9. Lento plot from 100 bootstrap trees of the woodmouse data

White et al. [312] proposed an alternative way to partition the phylogenetic signal present in a sequence alignment into three components—internal, terminal, and residual—summing to one and so can be represented as frequencies on a triangle. This has the advantage of displaying the phylogenetic information with a single point, thus several data sets (or subsets) can be represented simultaneously.

Finally, the bootstrap seems ideally suited to examine bias in the estimation of other evolutionary parameters (substitution rates, shape parameter of inter-site variation, branch lengths, ...) though this does not seem to have attracted some attention (see Freedman [91] for the use of the bootstrap to assess estimator bias).

5.5.3 Distances Between Trees

The idea of distances between trees is related to the bootstrap because this requires summarizing and quantifying the variation in topology among different trees. Several ways to compute these distances have been proposed in the literature [79, Chap. 30]. Two of them are available in the function `dist.topo`.

Penny and Hendy [241] proposed measuring the distance between two trees as twice the number of internal branches that differ in their splits. Rzhestky and Nei [272] proposed a modification of this distance to take multichotomies into account: this is the default method in `dist.topo`. For a trivial example:

```
> tr <- read.tree(text = "((a,b),(c,d));")
> tb <- read.tree(text = "((a,d),(c,b));")
> dist.topo(tr, tb)
[1] 2
> dist.topo(tr, tr)
[1] 0
```

Kuhner and Felsenstein [170] proposed to calculate the square root of the sum of the squared differences of the internal branch lengths defining similar splits in both trees. They called this distance as the branch length score; it is somewhat similar to the previous distance but taking branch lengths into account.

```
> tr <- rtree(10)
> tb <- rtree(10)
> dist.topo(tr, tb, "score")
[1] 2.270937
```

The function `treedist` in phangorn offers an alternative implementation by computing four distances for a pair of trees:

```
> treedist(tr, tb)
     symmetric.difference   branch.score.difference
                14.000000                  2.587576
```

```
path.difference quadratic.path.difference
      16.248077                 10.082005
```

The "symmetric difference" is the same than Penny and Hendy's distance in `dist.topo`. The "path difference" is the difference in path lengths, counted as the numbers of branches, between the pairs of tips [290], while the "quadratic path difference" is the same but using branch lengths.

Billera, Holmes, and Vogtmann [23] developed an elaborate distance based on the concept of tree space. This space is actually a cube complex because it is made up of cubes that share certain faces. Two trees with the same topology lie in the same cube of dimension $n-2$ (n being the number of tips). If they do not have the same topology they will be in two distinct cubes. However, these cubes meet at the origin where the internal branches that are different between the trees are equal to zero (Fig. 5.10). Thus it is possible to define a distance between two rooted trees across these interconnected cubes which is called a geodesic distance. The package distory implements this distance as well as other tools. The function `dist.multiPhylo` calculates the geodesic given a list of trees, which could be an object of class `"multiPhylo"`, and returns an object of class `"dist"`. For instance, computing the distance between the two binary trees on Fig. 5.10:

```
> library(distory)
> ta <- read.tree(text = "((a:1,b:1):1,c:1);")
> tc <- read.tree(text = "((a:1,c:1):1,b:1);")
> dist.multiPhylo(c(ta, tc))
  1
2 2
```

For rooted trees `phylo.diff` is a graphical tool to identify internal branches that are unique between two trees (see also `compare.phylo`, p. 54).

The fact that the geodesic distance has a physical interpretation leads to the possibility of using it in some applications related to the idea of a tree space. distory implements some of them. For instance, the help page `?dist.multiPhylo` gives an example of the use of the geodesic distance together with multidimensional scaling (also known as principal coordinates analysis) to represent geometrically a set of bootstrap trees.

Another application is provided by the function `mcmc.target.seq` which finds, using MCMC, a bootstrap sample of a `"DNAbin"` object that is closest to a target tree in terms of geodesic distance [68]. The distance at each step is output so that the convergence of Markov chain can be assessed.

5.5.4 Consensus Trees and Networks

Consensus trees are an interesting way to summarize a set of trees: if they are dichotomous, the clades not observed in all (strict consensus) or the majority (majority-rule consensus) will be collapsed as multichotomies.

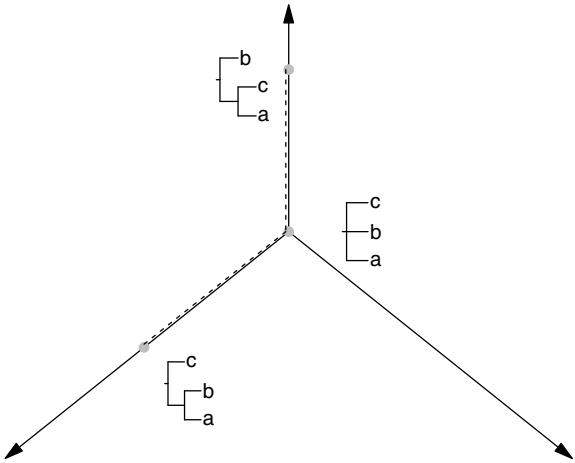

Fig. 5.10. The tree space for $n = 3$. It is made of three 1-d spaces interconnected at their origin—and not the three dimensions of a 3-d space. The grey dots indicate the positions of the three trees in this space. The dashed line shows the geodesic distance between the two binary trees. In addition each tree is characterized by an n-d Euclidean space representing the terminal branches

The function `consensus` in `ape` returns the consensus from a list of trees given in the same way as for `prop.part` or `prop.clades`. There is one option, `p`, which specifies the threshold, as a number between 0.5 and 1, of the proportion of the splits for their inclusion in the consensus tree. If `p = 1` (the default), then the strict consensus tree is returned, whereas `p = 0.5` returns the majority-rule consensus tree. This corresponds to the parameter l of the M_l consensus methods in [79].

This parameter l cannot be smaller than 0.5 because the splits not observed in at least 50% of the trees are incompatible with some observed in the majority of the trees and so cannot be represented together in the same tree. A way to solve this problem is to display alternative branchings with reticulations. Consider the unrooted trees in the middle of Fig. 5.11: their internal branches define splits, namely 12|34, 13|24, and 14|23, which clearly cannot be observed in the same tree and thus are all incompatible. The network on the top left represents simultaneously the two first trees with a rectangle symbolizing the two internal branches. In order to represent all three splits, we need a cube as shown by the network on the right of Fig. 5.11.

The remarkable thing about consensus networks is that once boxes and rectangles have been identified like in Fig. 5.11, they can be assembled using the geometry of some structures known as squaregraphs [14]. Figure 5.12 shows an example of a network made of two rectangles displaying three splits. An internal branch in an unrooted tree is represented as two or more parallel

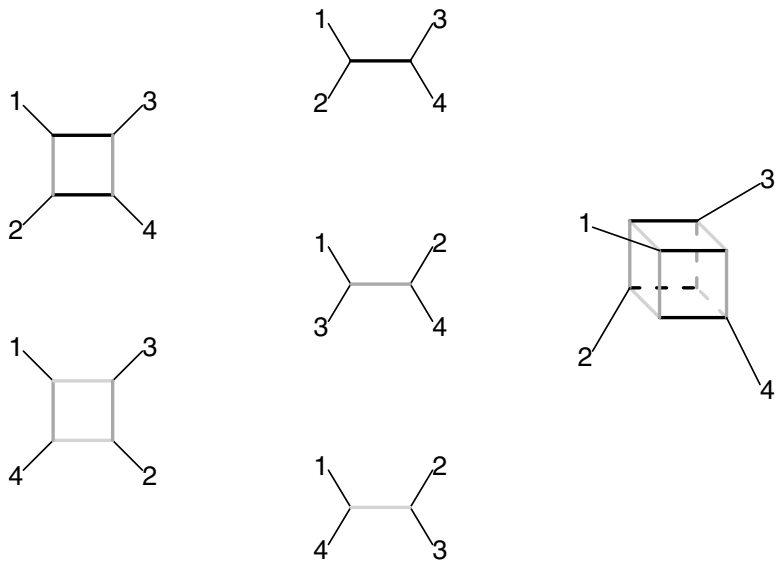

Fig. 5.11. The three possible unrooted topologies with $n = 4$ (middle), two out of the three possible consensus networks involving two topologies (left), and the consensus network representing the three topologies (right). The internal branches and their representations in the networks are shown with the same shades of grey

edges in a consensus network. This idea was originally introduced by Bandelt [15, 16, 17].

The function `consensusNet` in `phangorn` computes the consensus network for a list of trees given as first argument or, equivalently, a list of splits. The second argument, `prob`, gives the minimum frequency for a split to be uncluded in the network (0.33 by default). We come back to the bootstrap trees from the woodmouse data:

```
> cnt <- consensusNet(s) # same than consensusNet(o$trees)
> class(cnt)
[1] "networx" "phylo"
> cnt

Phylogenetic tree with 15 tips and 17 internal nodes.

Tip labels:
No305, No304, No306, No0906S, No0908S, No0909S, ...

Unrooted; includes branch lengths.
```

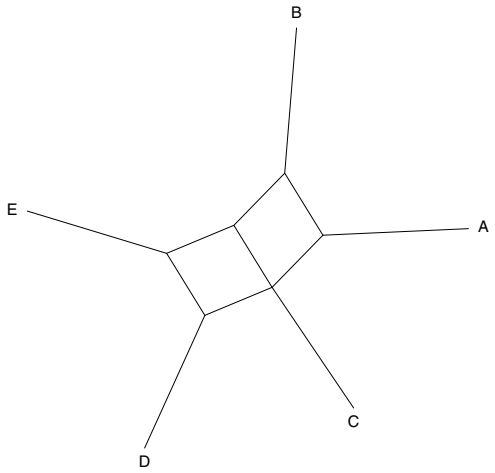

Fig. 5.12. A consensus network showing the splits AB|CDE, ABC|DE, and ACD|BE which is incompatible with the two previous ones

We note that the number of nodes is greater than the number of tips—in a tree without reticulation the former is always smaller than the latter. The network is then plotted with the appropriate method (Fig. 5.13):

```
plot(cnt, "2", tip.color = "black", edge.width = 1, font = 1)
```

The edge lengths are proportional to the frequencies of the splits, so a more "open" box indicates, in the present case, a stronger bootstrap support. This simple example clearly shows the advantage of the consensus network over a tree with bootstrap values. Here, the alternative positions of No1206S and No0908S cannot be represented together on the same tree, and similarly for No1007S, No1208S, and No0909S. Setting `prob = 0.1` gives logically a more complex network which is best visualized in 3-D with rgl (see Exercices).

The 3-D plot of consensus networks was originally developed by Schliep for phangorn [278]; the 2-D version used here is derived from this one (Klaus Schliep, personal communication). Other details on plotting networks can be found on page 117.

5.6 Molecular Dating

Branch lengths and substitution rates are confounded in phylogenetic estimation, therefore it is not possible to estimate branch lengths in units that are proportional to time. This must be done using additional assumptions on rate variations. This is the approach used in molecular dating methods. These techniques have two goals: transform a non-ultrametric tree into an

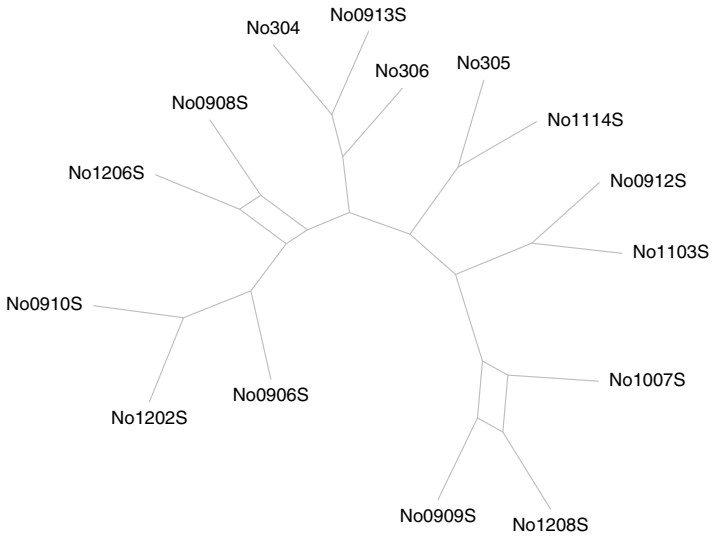

Fig. 5.13. Consensus network made from 100 bootstrap trees of the woodmouse data

ultrametric one, and estimate the absolute dates of the nodes of a tree. The latter requires at least one "calibration point", that is a node of the phylogeny which date is known. Each of these dates may be known exactly or within an interval defined by a lower (younger) and upper (older) bounds.

5.6.1 Molecular Clock

If substitution rate is constant, then molecular sequences are said to evolve according to the molecular clock: molecular divergence is related to time. Note that even if the substitution rate is constant, we do not expect a strictly linear relationship between molecular distance and time because of the stochastic nature of molecular evolution [163].

There are various methods in the literature to estimate a dated tree under a molecular clock model [79]. Britton et al. [32] proposed the mean path lenghts method where the date of a node is estimated with the mean of the distances from this node to all tips descending from it. This is implemented in the function `chronoMPL` in `ape` which takes as argument a phylogeny whose branch lengths are the mean numbers of substitutions, so branch lengths in numbers of susbtitutions per site must be multiplied by the sequence length before analysis. Under the molecular clock assumption, the two subtrees originating from a node are expected to accumulate similar molecular divergence: this leads to a test, computed by default by `chronoMPL`, for each node of the phylogeny. The results are returned as attributes of the dated tree (and thus extracted with `attr`; see `?chronoMPL` for an example).

The present method must be seen as mainly exploratory. Often, a plot of the original rooted phylogeny is very informative on variation in substitution rate.

5.6.2 Penalized Likelihood

Sanderson [275] proposed a method called nonparametric rate smoothing (NPRS) which assumes that each branch of the tree has its own rate, but these rates change smoothly between connected branches. Given a tree with estimated branch lengths in terms of number of substitutions, it is possible to estimate the dates of the nodes by minimizing the changes in rates from one branch to another. Practically this is done by minimizing the function:

$$\Phi = \sum |\hat{r}_k - \hat{r}_j|^p \ , \tag{5.12}$$

where $\hat{r}$ is the estimated substitution rate, k and j are two nodes of the same branch, and p is an exponent (usually 2). In order to make a trade-off between clock-like and non-clock models, Sanderson [276] proposed to modify his method by using a semiparametric approach based on a penalized likelihood. The latter (denoted Ψ) is made of the likelihood L of the "saturated" model (the one that assumes one rate for each branch of the tree) minus a roughness penalty which is similar to (5.12) multiplied by a smoothing parameter λ:

$$\Psi = \ln L - \lambda\Phi \ , \tag{5.13}$$

$$L = \prod r^x \frac{\mathrm{e}^{-r}}{x!} \ . \tag{5.14}$$

with x being the number of substitutions observed on a branch. The product in L is made over all branches of the tree. If $\lambda = 0$ then the above model is the saturated model with one distinct rate for each branch. If $\lambda = +\infty$, then the model converges to a clock-like model with the same rate for all branches. In order to choose an optimal value for λ, Sanderson [276] suggested a cross-validation technique where each terminal branch is removed from the data and then its length is predicted from the reduced data set. A different criterion is used here:

$$D_i^2 = \sum_{j=1}^{n-2} \frac{(\hat{t}_j - \hat{t}_j^{-i})^2}{\hat{t}_j} \ , \tag{5.15}$$

where $\hat{t}_j$ is the estimated date for node j with the full data, and $\hat{t}_j^{-i}$ is the one estimated after removing tip i. This criterion is easier to calculate than Sanderson's [276].

The penalized likelihood method is implemented in the function `chronopl`; its interface is:

```
chronopl(phy, lambda, age.min = 1, age.max = NULL,
         node = "root", S = 1, tol = 1e-8, CV = FALSE,
         eval.max = 500, iter.max = 500, ...)
```

where `phy` is an object of class `"phylo"` with branch lengths giving the number of substitutions (or its expectation), `lambda` is the smoothing parameter λ, `age.min` and `age.max` are numeric vectors giving the lower and upper bounds of the dates that are known, `node` is the number of the nodes whose dates are known (the last three arguments must be of the same length), `S` is the number of sites used as unit for the original branch lengths (usually this must be left as default), `CV` is a logical specifying whether to do the cross-validation, and `tol`, `eval.max` and `iter.max` are usual parameters to control the estimation process. This function returns a tree with branch lengths proportional to time (i.e., a chronogram) with attributes `rates` (the estimated absolute rates, $\hat{r}$), and `ploglik` (the penalized likelihood). If `CV = TRUE`, an additional attribute `D2` is returned with the value calculated with (5.15) for each tip.

The cross-validation may be done for different values of λ, for instance, for $\lambda = 0.1, 1, 10, \ldots, 10^6$:

```
l <- 10^(-1:6)
cv <- sapply(l, function(x) sum(attr(chronopl(phy,
                    lambda = x, CV = TRUE), "D2")))
plot(l, cv)
```

Sanderson suggested selecting the value of λ that minimizes the cross-validation criterion. If `CV = TRUE`, `chronopl` returns a value D_i^2 for each tip, so it is possible to examine which observations are particularly influential, for instance with:

```
chr <- chronopl(phy = phy.est, lambda = 1, CV = TRUE)
plot(attr(chr, "D2"), type = "l")
```

5.6.3 Bayesian Dating Methods

A Bayesian approach to molecular dating requires to define priors for the substitution rates and for the dates of the tree. There are several ways to do the latter, the most common one being to assume that the times between branching events follow a specific distribution. Thorne et al. [300] proposed to use a Dirichlet distribution (a Dirichlet process generates multidimensional variables that sum to one). Other methods are based on a birth-death process [317]. Defining a prior for the substitution rate is somewhat easier and a lognormal distribution is usually assumed [see 300, for details].

As mentioned above, implementing Bayesian methods requires a lot of programming and it makes sense to interface existing programs with R. That is what the package LAGOPUS does with its function `multidivtime` which calls Thorne's programs 'estbranches' and 'multidivtime' which themselves

call Yang's program 'baseml'. These programs must therefore be installed on the computer and accessible from R's working directory. The most efficient way to achieve this is to create symbolic links from the executable binaries to the working directory. Running multidivtime is quite tedious and LAGOPUS makes a very good job to have the results returned into R.

The input data is an object of class `"mdt.in"` built with the function of the same name: it includes all the sets of sequences, associated trees, and a data frame defining the time calibration points. The list `multicntrl.dat` controls the MCMC run by multidivtime and must be loaded into R:

```
> library(LAGOPUS)
> data(multicntrl.dat)
> multicntrl.dat
$numsamps
[1] 10000

$sampfreq
[1] 100

$burnin
[1] 1e+05

$rttm
[1] 90

$rttmsd
[1] 90

$rtrate
[1] "median"

$rtratesd
[1] "median"

$brownmean
[1] 0.4

$brownsd
[1] 0.4

$minab
[1] 1

$newk
[1] 1
```

```
$othk
[1] 0.5

$thek
[1] 0.5

$bigtime
[1] 200

$tipsnotcoll
[1] 0

$nodata
[1] 0

$commonbrown
[1] 0
```

Once the data have been prepared with `mdt.in` and the above list has been loaded, `multidivtime` may be called in R; the inteface is:

```
multidivtime(x, file = NULL, start = "baseml", part = 1,
             runs = 1, path = NULL, transfer.files = TRUE,
             LogLCheck = 100, plot = TRUE)
```

The object returned is of class `"mdt"` and is a list with three components: `TREE` (the estimated chronogram), `TABLE` (the estimated dates with confidence intervals), and `CONSTRAINTS` (a recap of the time constraints used as calibration points). There is also a nice `plot` method which draws the credibility intervals computed from the posterior density of the dates over the estimated chronogram; this has many options.

Running `multidivtime` from LAGOPUS takes a few minutes with the default controls. So this package not only eases the use of an existing program, but also makes possible to repeat the analyses since Bayesian computations are relatively fast in the present case.

5.7 Summary and Recommendations

Figure 5.14 summarizes a workflow of phylogeny estimation derived from this chapter. This shows how the techniques exposed in the previous sections can be used for phylogenetic inference. This considers DNA sequences as the raw data: they are not the only kind of data that can be analyzed for estimating phylogenies, but they have appeared more and more clearly over the last few years as the most appropriate for this task [31, 64]. Table 5.5 gives for each

step, the points which are particularly important to examine, though they are not limitative.

There has been an increasing trend in the literature over the past few years to perform estimation of phylogenies as a black-box computation where data are input and a tree is output. The availability of high performance hardware and software, not surprisingly, has reinforced this practice. I defend here a different approach. Phylogeny estimation should be done like other statistical inference procedures. Thus, some attention must be paid to heterogeneity and variability in the processes under investigation. This obviously requires a combination of exploratory, graphical, and modeling tools. The description of such tools in this chapter, and these final recommendations, do not mean to be exhaustive in this respect. Nevertheless, I hope they will contribute to a more critical approach to phylogeny estimation.

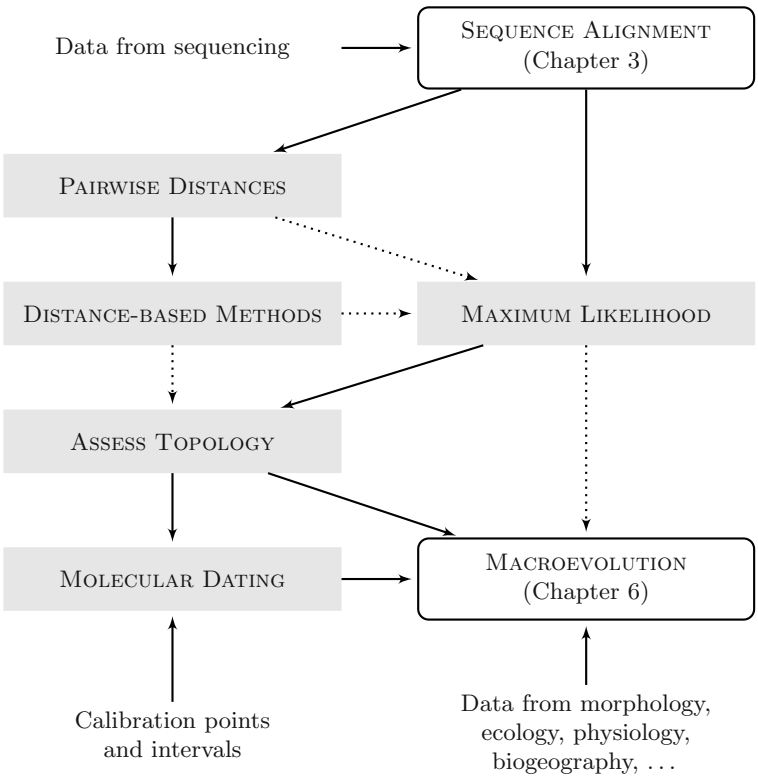

Fig. 5.14. Flowchart of phylogeny estimation. Straight arrows indicate direct flow of data; dotted arrows show when some results from the origin would impact the process in the destination

Table 5.5. Some points to be considered in phylogeny estimation

	Points to be examined	Tools
Pairwise Distances	Low level of variability	Histogram and summary statistics
	Multiple substitutions	Plot JC69 against uncorrected
	Ts/Tv bias (etc.)	Plot K80 against JC69, F81 against F84 (etc.)
	Heterogeneity along the sequences	Same than above with data subsets (e.g., codon positions)
	Heterogeneity among genes	Same than above with different plots on the same figure
	Saturation	Histogram and saturation plots
Dis. Met.	Robustness to the algorithm	Use different methods (NJ, FastME, BIONJ, ...)
	Impact of heterogeneity within / among sequences	Repeat analyses for different data subsets
Max. Likelih.	Test hypotheses on variation in substitution rate(s)	Fit alternative models
	– along sequences	With(out) Γ variation
	– among genes	With(out) partitions
	– parameters	Alternative models (GTR, ...)
	– base frequencies	Id.
	Alternative topologies	Shimodaira–Hasegawa test
Assess Top.	Confidence in the inferred topology	Bootstrap, MCMC
	Bias and conflicting signal	Lento plot, consensus network
		Geodesic distance with multidimensional scaling
	Incomplete lineage sorting	Species tree estimation from gene trees
Mol. Dating	Heterogeneity in substitution rate among branches	Plot rooted trees with scale
	Assess unclock-like of the substitution rate	Find the best value of λ with PL
	Impact of individual sequences	Examine individual contributions to CV with D_i^2 with PL

5.8 Case Studies

In this section, we come back to some of the data prepared in Chapter 3. We see how we can estimate phylogenies, eventually repeat some analyses done in the original publications, and possibly see how we could go further with R.

5.8.1 *Sylvia* Warblers

To continue with the *Sylvia* data, it may be necessary to reload the data prepared and saved previously:

```
load("sylvia.RData")
```

A distance matrix can be estimated from these aligned sequences using `dist.dna`; because 2 of the 25 sequences are substantially incomplete, we use the option `pairwise.deletion = TRUE`:

```
syl.K80 <- dist.dna(sylvia.seq.ali, pairwise.deletion = TRUE)
```

We recall that the default model for this function is Kimura's two-parameter one. We use the option `model` to try different models:

```
syl.F84 <- dist.dna(sylvia.seq.ali, model = "F84", p = TRUE)
syl.TN93 <- dist.dna(sylvia.seq.ali, model = "TN93", p = TRUE)
syl.GG95 <- dist.dna(sylvia.seq.ali, model = "GG95", p = TRUE)
```

A way to compare these distance matrices is simply to look at their correlations. We do this by binding all distances in a single matrix, and compute the correlations among its columns (the results are rounded to three digits):

```
> round(cor(cbind(syl.K80, syl.F84, syl.TN93, syl.GG95)), 3)
         syl.K80 syl.F84 syl.TN93 syl.GG95
syl.K80    1.000   1.000    1.000    0.928
syl.F84    1.000   1.000    1.000    0.927
syl.TN93   1.000   1.000    1.000    0.925
syl.GG95   0.928   0.927    0.925    1.000
```

This shows that the GG95 distances differ substantially from the others. Note that a perfect correlation does not guarantee that the distances are the same: some graphical analyses are needed to check this. We do this to examine the saturation of substitutions in the sequences. We first compute the distances using the JC69 model and the raw distance (i.e., proportion of different sites):

```
syl.JC69 <- dist.dna(sylvia.seq.ali, model = "JC69", p=TRUE)
syl.raw <- dist.dna(sylvia.seq.ali, model = "raw", p=TRUE)
```

We then plot these two distances expecting the raw distances to be smaller because they do not consider multiple substitutions; we also plot the Jukes–Cantor distance *versus* the Kimura one to show the potential influence of the transition/transversion ratio (Fig. 5.15):

```
layout(matrix(1:2, 1))
plot(syl.JC69, syl.raw); abline(b = 1, a = 0) # draw x=y line
plot(syl.K80, syl.JC69); abline(b = 1, a = 0)
```

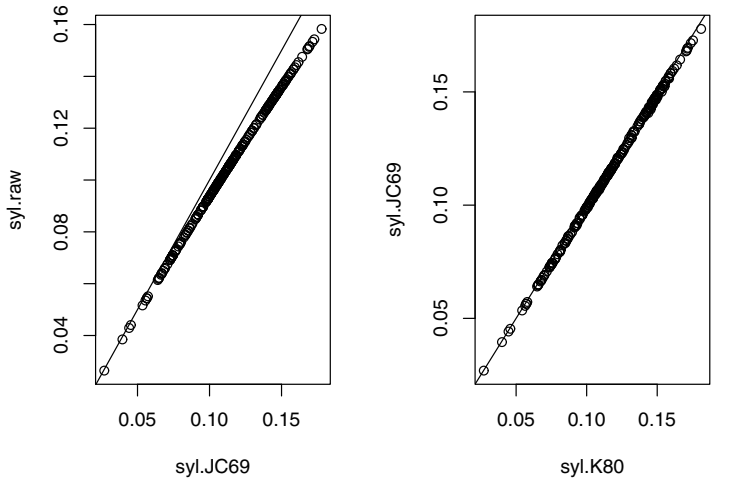

Fig. 5.15. Saturation plots for the cytochrome *b* sequences of 25 species of *Sylvia* showing the effects of multiple substitutions (left) and of the transition/transversion ratio (right)

These plots show, as expected, that the most divergent sequences are slightly saturated, whereas the transition/transversion ratio does not seem to affect the estimated distances greatly.

This analysis, though informative, is not what is usually called "saturation plots" in the literature. The latter is a plot, eventually for each codon position, of the numbers of transitions and transversions on the x-axis against the K80 distance on the y-axis. These graphs are not shown here but the code to do them is as follows:

```
layout(matrix(1:3, 1))
for (i in 1:3) {
    s <- logical(3); s[i] <- TRUE
    x <- sylvia.seq.ali[, s]
    d <- dist.dna(x, p = TRUE)
    ts <- dist.dna(x, "Ts", p = TRUE)
    tv <- dist.dna(x, "Tv", p = TRUE)
    plot(ts, d, xlab = "Number of Ts or Tv", col = "blue",
         ylab = "K80 distance", xlim = range(c(ts, tv)),
         main = paste("Position", i))
    points(tv, d, col = "red")
}
```

A special attention must be paid to the scales of the x- and y-axes when interpreting the three plots created by these commands.

A complementary graphical analysis can be done by plotting a histogram of the pairwise distances for each codon position (as suggested in Table 5.5). In order to plot the three histograms on the same scales, we use the function `histogram` in lattice instead of the usual `hist`. To this end, we prepare the data by computing the distances for each position, and concatenating them in a single vector. We then create a factor variable so that `histogram` can attribute each value of the concatenated vector to its codon position which is used as a conditional variable in the call of this function (Fig. 5.16):

```
y <- numeric()
for (i in 1:3) {
    s <- logical(3); s[i] <- TRUE
    y <- c(y, dist.dna(sylvia.seq.ali[, s], p = TRUE))
}
g <- gl(3, length(y) / 3)
library(lattice)
histogram(~ y | g, breaks = 20)
```

The figure is very revealing: not only the mean pairwise distance is affected by the codon position but also its variance. This will have a significant impact on the choice of model for phylogeny estimation.

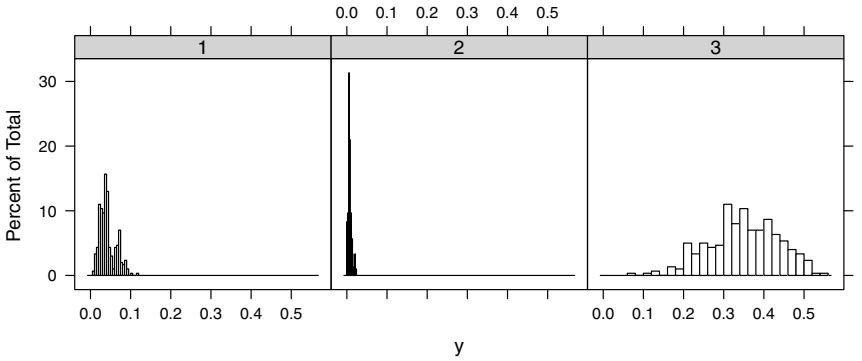

Fig. 5.16. Pairwise histograms for the cytochrome b sequences of 25 species of *Sylvia* for the three codon positions

A point we explore briefly is the impact of the choice of the substitution model on the phylogeny estimation with the NJ method. We estimate a tree with the function `nj` for the K80 and GG95 distance matrices:

```
nj.sylvia.K80 <- nj(syl.K80)
```

```
nj.sylvia.GG95 <- nj(syl.GG95)
```

To see if the estimated topologies are similar, we compute the topological distance between them:

```
> dist.topo(nj.sylvia.K80, nj.sylvia.GG95)
[1] 12
```

We now do a bootstrap analysis like the one reported by Böhning-Gaese et al. [27] using `boot.phylo`. There are several ways to conduct this analysis. We choose to treat rooted trees, and so first identify the outgroup species, *Chamaea fasciata*:

```
> grep("Chamaea", taxa.sylvia, value = TRUE)
          AJ534526
"Chamaea_fasciata"
```

We then build a function that includes the `root` function in order to estimate the NJ tree including the rooting operation:

```
f <- function(xx) root(nj(dist.dna(xx, p=TRUE)), "AJ534526")
tr <- f(sylvia.seq.ali)
## same than: tr <- root(nj.sylvia.K80, "AJ534526")
```

We may now call `boot.phylo` specifying the option `rooted = TRUE` and using 200 bootstrap replicates as in [27]:

```
> nj.boot.sylvia <- boot.phylo(tr, sylvia.seq.ali, f, 200,
+                              rooted = TRUE)
> nj.boot.sylvia
 [1] 200 186  78 143 146 103 107 200 187  94 192 197  33 154
[15]  82  86 170 195  86 197 134 193  72
```

How could these bootstrap values have been influenced by the fact that we deal with coding sequences? We can assess this by using the option `block` of `boot.phylo`; this will result in resampling at the codon level instead of at the site level:

```
> nj.boot.codon <- boot.phylo(tr, sylvia.seq.ali, f, 200, 3,
+                             rooted = TRUE)
> nj.boot.codon
 [1] 200 193  72 139 134  99  99 198 180  87 192 194  38 163
[15]  75  87 164 194  97 196 140 196  94
```

The results are very close to the site-level resampling analysis; we thus consider the latter in the following.

We now plot the estimated tree by NJ with the bootstrap values on the nodes. We first copy the estimated tree and substitute the accession numbers (which were used as tip labels) with the species names:

```
nj.est <- tr
nj.est$tip.label <- taxa.sylvia[tr$tip.label]
```

The tree is then plotted with `plot`, the bootstrap values are added with `nodelabels`, and we draw a scale bar (Fig. 5.17):

```
plot(nj.est, no.margin = TRUE)
nodelabels(round(nj.boot.sylvia / 200, 2), bg = "white")
add.scale.bar(length = 0.01)
```

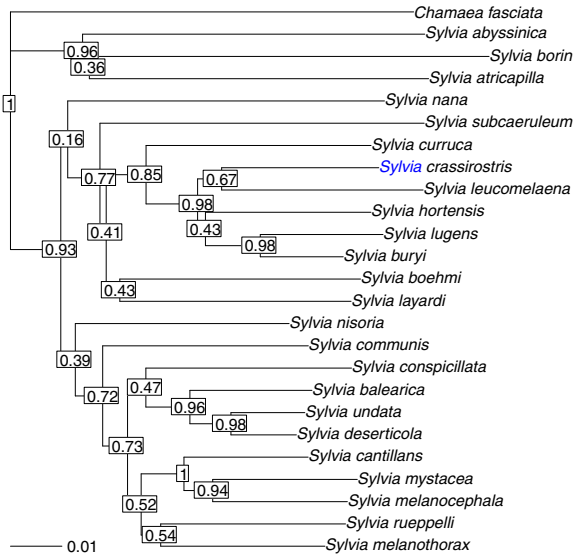

Fig. 5.17. Phylogenetic relationships among 25 species of the genus *Sylvia* based on cytochrome *b* sequences analyzed with neighbor-joining and Kimura's two-parameter distance

The bootstrap values shown in Fig. 5.17 are very close to those obtained by Böhning-Gaese et al. [27]. We finally save the final tree in a file using the Newick format:

```
write.tree(nj.est, "sylvia_nj_k80.tre")
```

We now move on to maximum likelihood by first performing an analysis with `phymltest`. PhyML has been installed and set as described on page 159. The command for the present analysis is thus simply (after writing the sequences in a file):

```
write.dna(sylvia.seq.ali, "sylvia.txt")
phyml.sylvia <- phymltest("sylvia.txt", execname = "~/phyml")
```

This takes a few minutes to run on a modern laptop. Displaying the results shows the log-likelihood and AIC values for each model:

```
> phyml.sylvia
          nb.free.para    loglik      AIC
JC69                 1 -9022.097 18046.19
JC69+I               2 -8307.898 16619.80
JC69+G               2 -8227.001 16458.00
JC69+I+G             3 -8223.574 16453.15
K80                  2 -8508.629 17021.26
K80+I                3 -7766.568 15539.14
K80+G                3 -7680.375 15366.75
K80+I+G              4 -7656.458 15320.92
F81                  4 -8941.140 17890.28
F81+I                5 -8157.509 16325.02
F81+G                5 -8072.674 16155.35
F81+I+G              6 -8055.982 16123.96
F84                  5 -8430.607 16871.21
F84+I                6 -7575.661 15163.32
F84+G                6 -7489.272 14990.54
F84+I+G              7 -7467.842 14949.68
HKY85                5 -8471.269 16952.54
HKY85+I              6 -7600.758 15213.52
HKY85+G              6 -7505.759 15023.52
HKY85+I+G            7 -7482.774 14979.55
TN93                 6 -8413.108 16838.22
TN93+I               7 -7569.273 15152.55
TN93+G               7 -7483.681 14981.36
TN93+I+G             8 -7461.591 14939.18
GTR                  9 -8350.477 16718.95
GTR+I               10 -7548.011 15116.02
GTR+G               10 -7467.290 14954.58
GTR+I+G             11 -7444.848 14911.70
```

The function `summary` computes all possible paired likelihood ratio tests (211 tests):

```
> summary(phyml.sylvia)
       model1    model2       chi2 df  P.val
1        JC69    JC69+I 1428.39644  1 0.0000
2        JC69    JC69+G 1590.19116  1 0.0000
3        JC69  JC69+I+G 1597.04450  2 0.0000
4        JC69       K80 1026.93474  1 0.0000
5        JC69     K80+I 2511.05698  2 0.0000
....
```

We can plot these results to have a more synthetic view (Fig. 5.18):

```
plot(phyml.sylvia, col = "black")
```

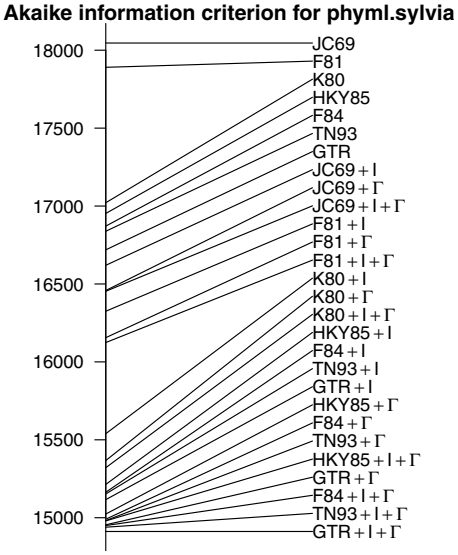

Fig. 5.18. Results of the analysis of 25 species of the genus *Sylvia* based on cytochrome *b* sequences with `phymltest`

The most complex model GTR + I + Γ is the one that best explains the data in terms of AIC. An interesting pattern from Fig. 5.18 is that for a given substitution model, adding invariants (I) considerably improves the fit, whereas this improvement is even better by adding Γ, and again better with both; thus there is a hierarchy $X >>> X + I >> X + \Gamma > X + I + \Gamma$ (X being a substitution model).

When comparing the substitution models, the key element seems to take the transition / transversion ratio into account. Once this has been included in the model (F80 being the simplest one), taking unequal base frequencies into account is also important although less than the previous parameter.

Once the analysis with `phymltest` has been done, it is possible to read the trees estimated by PhyML:

```
> TR <- read.tree("sylvia.txt_phyml_tree.txt")
> TR
28 phylogenetic trees
```

This file contains the 28 trees estimated by PhyML, the last one being the one estimated with the most complex model. We extract this tree, substitute its tip labels to get the species names in place of the accession numbers, and root the tree with *Chamaea fasciata* as outgroup (Fig. 5.19):

```
mltr.sylvia <- TR[[28]]
mltr.sylvia$tip.label <- taxa.sylvia[mltr.sylvia$tip.label]
mltr.sylvia <- root(mltr.sylvia, "Chamaea_fasciata")
plot(mltr.sylvia, no.margin = TRUE)
add.scale.bar(length = 0.01)
```

Naturally, the next step of our analyses should be to fit partitioned models because of the heterogeneity we have characterized in relation to codon position. However, the practical details of this approach have been exposed with the woodmouse data earlier in this chapter (p. 154), and we shall keep the present analysis shorter than it should be.

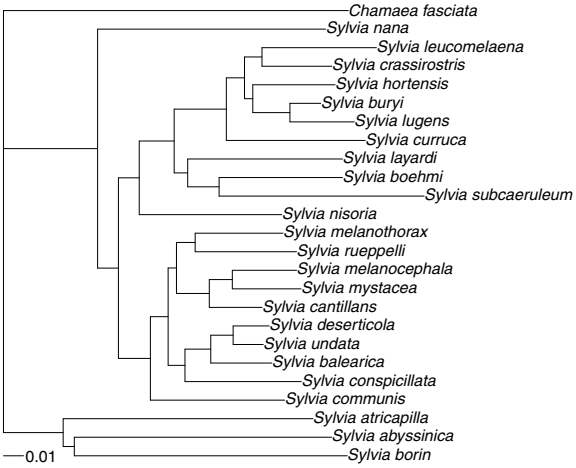

Fig. 5.19. Maximum likelihood phylogeny estimate of 25 species of the genus *Sylvia* based on cytochrome *b* sequences with the GTR + I + Γ model

From this ML estimate of the phylogeny of *Sylvia*, we can now estimate a chronogram with the penalized likelihood method [276]. The fossil record of these birds is extremely sparse and it is difficult to obtain a calibration point from it. Blondel, Catzeflis and Perret [26] found, using a molecular clock, a time of origin of the common ancestor of *Sylvia* between 12.1 and 13 million years ago which, unsurprisingly, agrees with the date estimated by Sibley and Ahlquist [284]. For the present analysis, we use an interval 12-16 Ma for the most recent common ancestor of *Sylvia*. We perform the penalized likelihood analysis, with cross-validation, with values of the smoothing parameter (λ) $10^{-4}, 10^{-3}, \ldots, 10^{3}, 10^{4}$. Before proceeding, we drop the outgroup from the estimated maximum likelihood tree:

```
tr.ml <- drop.tip(mltr.sylvia, "Chamaea_fasciata")
res <- vector("list", 9)
```

```
for (L in -4:4)
    res[[L + 5]] <- chronopl(tr.ml, 10^L, 12, 16, CV = TRUE)
```

We now plot the sum of the cross-validation values (5.15) against the value of λ (Fig. 5.20):

```
Lambda <- 10^(-4:4)
CV <- sapply(res, function(x) sum(attr(x, "D2")))
plot(Lambda, CV / 1e5, log = "x")
```

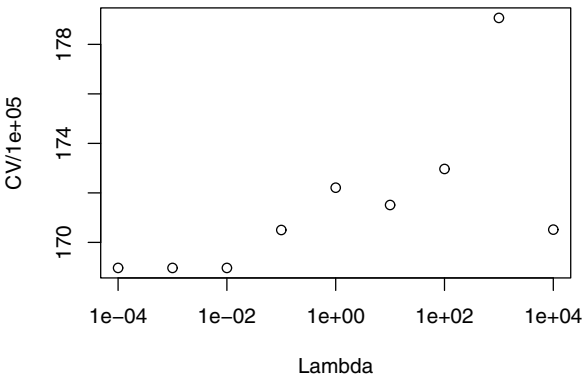

Fig. 5.20. Plot of the cross-validation criterion (CV) with respect to the value of the smoothing parameter λ

The lowest values show the lowest discrepancy between the predicted and the observed values. Therefore, we select the chronogram estimated with $\lambda = 10^{-3}$. We remind that a small value of λ implies heterogeneous substitution rates among brances (because we have set an absolute time frame, branch lengths and substitution rates can be disentangled). The attribute `"rates"` stores the estimated absolute rates of substitution:

```
> sylvia.chrono <- res[[2]]
> rts <- attr(sylvia.chrono, "rates")
> summary(rts)
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
0.002187 0.013950 0.025660 0.025350 0.034200 0.059810
```

These values are ordered along the edges of the tree, so they can be used as argument to some options of `plot.phylo`, such as `edge.color` or `edge.width`.

We now plot the estimated chronogram with the edge thickness proportional to these rates, and draw the time-axis with `axisPhylo` (Fig. 5.21). The scaling factor of 100 is found after several attempts in order to get the best visual effect:

```
par(mar = c(2, 0, 0, 0))
plot(sylvia.chrono, edge.width = 100*rts, label.offset = .15)
axisPhylo()
```

We finally save this chronogram for further analysis:

```
write.tree(sylvia.chrono, "sylvia.chrono.tre")
```

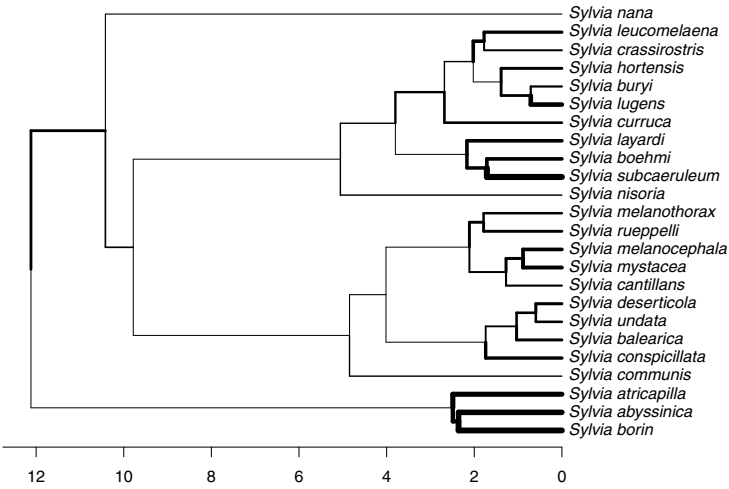

Fig. 5.21. Chronogram of the *Sylvia* warblers based on the penalized likelihood method. The branch thicknesses are proportional to the absolute rates of molecular evolution

5.8.2 Butterfly DNA Barcodes

We have 466 aligned sequences of COI: we limit ourselves here to simple analyses. Hebert et al. [131] showed that there seem to be several (ten actually) species instead of one originally recognized. We compute the pairwise distances between all specimens with `dist.dna`. We take care to use the option `pairwise.deletion = TRUE` because many sequences do not have the same length:

```
M.astraptes.K80 <- dist.dna(astraptes.seq.ali, p = TRUE)
```

We look at the distribution of the distances using `summary`:

```
> summary(M.astraptes.K80)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.00000 0.01590 0.02107 0.02749 0.03887 0.08326
```

We may plot an histogram of the 108,345 distances (Fig. 5.22):

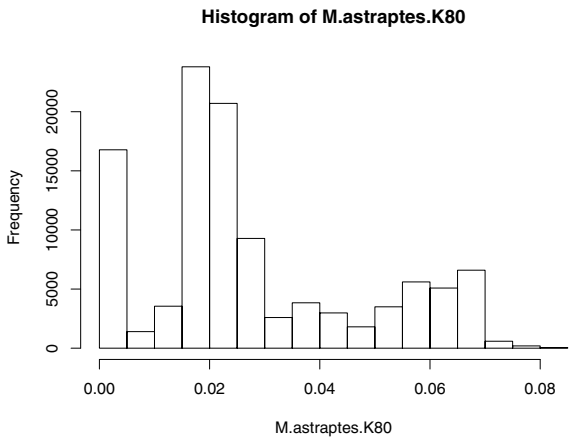

Fig. 5.22. Distribution of pairwise distances among 466 specimens *Astraptes fulgerator* based on cytochrome oxydase I sequences analyzed with Kimura's two-parameter distance

```
hist(M.astraptes.K80)
```

This clearly shows three peaks in the distribution: at 0, around 0.02, and around 0.07. This is in complete agreement with Hebert et al.'s results which showed that these peaks correspond to differentiation within populations, intraspecies, and interspecies, respectively.

It is possible to estimate an NJ tree with the distance matrix to assess how the different taxa are differentiated:

```
tr <- nj(M.astraptes.K80)
tr$tip.label <- taxa.astraptes[tr$tip.label]
```

The resulting tree is a bit too large to be displayed with `plot.phylo`, so we may use `zoom` instead. For this we have to find the indices of each taxon in the vector of tip labels. Here is a possible solution:

```
taxon <- unique(taxa.astraptes)
L <- vector(mode = "list", length = 10)
```

```
for (i in 1:10)
    L[[i]] <- grep(taxon[i], tr$tip.label)
```

We can now use `L` as an argument to `zoom`. We may plot all the subtrees at once in a large PDF file with:

```
pdf("astraptes.pdf", width = 30, height = 30)
zoom(tr, L)
dev.off()
```

and then open it with an appropriate viewer. Each taxon can be visualized separately with, for instance, `zoom(tr, L[1])`.

5.9 Exercises

1. (a) Show that ultrametric distances are also Euclidean.
   (b) Simulate some data with `rTraitCont` and show that distances among these variables may be Euclidean. Compare with data generated with `rnorm`.
   (c) Show that DNA distances cannot be Euclidean. Find a distance method with continuous variables that shows the same property. See the formula in `?dist` and compare with what you know on how DNA distances are calculated.

2. Compare the seven methods available in `upgma`. You will draw a single figure with the seven UPGMA trees and the necessary annotations. You will take a data set of your choice.

3. Longer distances inferred from molecular sequences tend to have higher variances than the shorter ones. Derive a weighted version of the least squares formula by modifying equation 5.1 (p. 136) in order to give less importance to longer distances. Find a diagnostic plot that will confirm the rationale of this weighting scheme. (Hint: you may type `example(lm)` in R to find some inspiration.)

4. Consider a DNA sequence that evolves according to the Jukes–Cantor (JC69) model.
   (a) Build the corresponding rate matrix using for the overall rate of change the value 3×10^{-4}.
   (b) Compute, using two different approaches, the probability matrix for $t = 1$, $t = 1000$, and $t = 1 \times 10^6$. What do you observe? Was that expected?
   (c) What could you conclude about phylogeny estimation from this exercise?

5. Consider a GTR model with the following parameters: $\alpha = 0.001$, $\beta = 5 \times 10^{-4}$, $\gamma = 2 \times 10^{-4}$, $\delta = 3 \times 10^{-4}$, $\epsilon = 1 \times 10^{-4}$, $\zeta = 5 \times 10^{-5}$, $\pi_A = 0.35$, $\pi_G = 0.17$, $\pi_C = 0.25$, and $\pi_T = 0.23$.
   (a) Build the corresponding rate matrix.
   (b) Compute the probability matrix for $t = 1$.
   (c) Find a method to simulate the evolution of a DNA sequence under this GTR model for an arbitrary t.
   (d) What are the expected base frequencies when t is very large?

6. Write R code to calculate the distance between two aligned nucleotide sequences with the GTR model using equation 5.8 (p. 144). (Hints: you will need the functions `base.freq`, `Ftab`, `eigen`, and `solve`. The 'trace' function is the sum of the diagonal elements of a matrix.)

7. (a) Give the R code to calculate `Py` in Section 5.2.2. Compare with `Px`. How this will affect subsequent calculations? Eventually try different values of α.
   (b) Why the likelihood values do not sum to one?
   (c) For site 2, why the likelihood for A is twice bigger than for T?
   (d) How many parameters are involved in these calculations?
   (e) Write an R function to calculate the likelihood of the (full) data; the arguments of this function will be these parameters.

8. Sketch a function doing Bayesian estimation of phylogeny. The code should include comments explaining the rationale of the choices.

9. Take the data prepared in Exercise 5 of Chapter 3.
   (a) Build saturation diagrams for the whole sequence, and for each codon position.
   (b) Examine graphically the effects of unequal transition and transversion rates and / or unequal base frequencies on the distance estimates for each data set (whole sequences and each codon position).

10. Generate 100 bootstrap trees from the woodmouse data (see p.176). Compute and plot the consensus network displaying the splits with frequency 0.1 or more. Compare with Fig. 5.13 and explain the differences. Repeat with 1000 bootstrap trees.