Definition of Formats for Coding Phylogenetic Trees in R

Emmanuel Paradis

December 4, 2006

Contents

1 Introduction

This document explains how phylogenetic trees are coded and handled in the R package ape. Such data, like any R data, can be passed to C codes for computing-intensive tasks. How these data can be manipulated efficiently in R, and how they can be extended is discussed.

Phylogenetic trees are complex data structures: coding them in computer programs requires special care. Felsenstein [2] outlines the coding used in most

computer programs for phylogenetic analyses. Such data structures cannot be used easily in an interactive way because they are based on the recursive use of pointers. In addition, they cannot be extended or modified without recompiling the whole program. In R a different approach is used based on simple data structures like matrices and vectors. The class `"hclust"`, used by the function `hclust` of the package `stats`, codes hierarchical clusters with a two-column matrix indicating the pairings of observations. To code phylogenetic trees in R, it was necessary to extend and modify this class substantially.

The class `"phylo"` introduced in `ape` in August 2002 has this aim. Its initial definition in version 0.1 of `ape` has been useful in some applications. These and some feed-back from users have pointed out to some weaknesses and limitations of this initial definition. I detail below the new definition of the class `"phylo"` introduced in `ape` 1.9 released in November 2006.

2 Terminology and Notations

branch:	edge (= vertex)
node:	internal node
tip:	terminal node (= leaf)
n:	number of tips
m:	number of nodes

3 Definition of the Class `"phylo"`

An object of class `"phylo"` is a list with, at least, the following mandatory elements:

1. A numeric matrix named `edge` with two columns and as many rows as there are branches in the tree;
2. A character vector of length n named `tip.label` with the labels of the tips;
3. A numeric value named `Nnode` giving the number of (internal) nodes;
4. An attribute `class` equal to `"phylo"`.

In the matrix `edge`, each branch is coded by the nodes it connects: tips are coded $1, \ldots, n$, and internal nodes are coded $n+1, \ldots, n+m$ ($n+1$ is the root). Both series are numbered with no gaps.

The matrix `edge` has the following properties:

- The first column has only values $> n$ (thus, values $\leq n$ appear only in the second column).
- All nodes appear in the first column at least twice.
- The number of occurrences of a node in the first column is related to the nature of the node: twice if it is dichotomous (i.e., of degree 3), three times if it is trichotomous (degree 4), and so on.

- All elements, except the root $n + 1$, appear once in the second column (only if the tree has no reticulation).

This representation is used for rooted and unrooted trees. For the latter, the position of the root is arbitrary.

The smallest tree of class `"phylo"` can be created in R with:

```
> tr <- list(edge = matrix(c(2, 1), 1, 2), tip.label = "a", Nnode = 1)
> class(tr) <- "phylo"
> str(tr)

List of 3
 $ edge     : num [1, 1:2] 2 1
 $ tip.label: chr "a"
 $ Nnode    : num 1
 - attr(*, "class")= chr "phylo"
```

The following elements are optional:

1. A numeric vector named `edge.length` with the branch lengths: this has as many values of the number of rows of `edge`;
2. A character vector of length m named `node.label` with the labels of the nodes;
3. A single numeric value named `root.edge` giving the length of the branch at the root.

There is a correspondence between these elements and the structure of the mandatory ones (e.g., the length of the ith branch `edge[i, ]` is given by `edge.length[i]`).

There are several substantial changes compared to the definition used until version 1.8-5 of `ape` [3]: the most important one is that `edge` was previously of mode character with a different coding (tips had positive numbers `"1"`, ..., `"n"`, and nodes negative numbers `"-1"`, ..., `"-m"`). Compatibily is provided by some functions in `ape` 1.9.

Note that this new definition still uses the S3 classes, though a switch to the S4 classes may be envisaged in the future.

3.1 Memory Requirements

The table below gives the sizes as given by `object.size()` of three phylogenies distributed with `ape` and often used in examples, and two random trees generated with `rtree`. The column labelled "old" gives the size in bytes with the old definition of the class `"phylo"`, and the one labelled "new" gives the size using the new definition.[1]

[1] These figures were obtained with R 2.3.0; a more recent version gave slightly different values (see Appendix).

tree	n	old	new	gain
bird.orders	23	5560	2816	1.97
bird.families	135	30,992	13,720	2.26
chiroptera	917	158,900	72,972	2.18
random tree	1000	216,396	88,596	2.44
"	10,000	2,160,396	880,596	2.45

The gain in terms of memory requirements is thus at least twice. Less than 1 Mb is needed to store a tree of 10,000 tips with the new definition. Given than most computers have nowadays at least 1 Gb of RAM, this should make a wide range of analyses feasible.

3.2 Guidelines for Creating the Matrix `edge`

In the matrix `edge`, each row represents a branch: the node in the first column is the origin of the branch, and the node or tip in the second column is its end. Note that for unrooted trees, this order is arbitrary (except for the terminal branches) because the position of the root is also arbitrary. This representation allows a lot of generalizations, such as multichotomies (here $n = 3, m = 1$) ...

```
> matrix(c(rep(4, 3), 1:3), 3, 2)
```

```
     [,1] [,2]
[1,]    4    1
[2,]    4    2
[3,]    4    3
```

... or reticulations ($n = 4, m = 3$):

```
> cbind(c(5, 6, 6, 6, 5, 7, 7), c(6, 1, 2, 7, 7, 3, 4))
```

```
     [,1] [,2]
[1,]    5    6
[2,]    6    1
[3,]    6    2
[4,]    6    7
[5,]    5    7
[6,]    7    3
[7,]    7    4
```

There is an arbitrary direction ($6 \rightarrow 7$), from the smallest number to the largest one. At the moment, it is still an open question whether these reticulations should be coded in the matrix `edge`, and should rather be stored in a distinct matrix to avoid confusion.

There is no mandatory order for the rows of `edge`, but they may be arranged in a way that is efficient for computation and manipulation. For instance, consider the tree in Newick format:

```
"((,),(,));"
```

Then the two following matrices are similar for `edge`:

```
> cbind(c(5, 6, 6, 5, 7, 7), c(6, 1, 2, 7, 3, 4))

     [,1] [,2]
[1,]    5    6
[2,]    6    1
[3,]    6    2
[4,]    5    7
[5,]    7    3
[6,]    7    4

> cbind(c(5, 5, 6, 6, 7, 7), c(6, 7, 1, 2, 3, 4))

     [,1] [,2]
[1,]    5    6
[2,]    5    7
[3,]    6    1
[4,]    6    2
[5,]    7    3
[6,]    7    4
```

In the first representation the branches are grouped *cladewise*, whereas in the second one the internal branches come first. Any order of the rows are valid with respect to the above definition. However, the cladewise order has an interesting feature: it is straightforward to find all the branches descendant of a given node (see § 4.2).

There is another interesting order:

```
> cbind(c(6, 6, 7, 7, 5, 5), c(1, 2, 3, 4, 6, 7))

     [,1] [,2]
[1,]    6    1
[2,]    6    2
[3,]    7    3
[4,]    7    4
[5,]    5    6
[6,]    5    7
```

Here, the branches are arranged so that a "pruning" calculation (or postorder tree traversal) can be done by reading down the rows of `edge`. Additionally, if some conventions are taken, this arrangement can lead to a unique representation for a given tree in the same way than the matchings proposed by Diaconis & Holmes [1].[2] I shall call the above order *pruningwise*.

4 Tree Manipulation

The table below shows how to perform a few basic operations on objects of class `"phylo"` in R.

[2] Matchings work only for rooted dichotomous trees.

How many tips?	`length(tr$tip.label)`
How many nodes?	`tr$Nnode`
How many branches?	`dim(tr$edge)[1]`
How to find node `x`?	`which(tr$edge == x)` *or* `which(tr$edge == x, TRUE)`
What is the ancestor of node `x`?	`i <- which(tr$edge[, 2] == x)` `tr$edge[i, 1]`
What are the terminal branches?	`n <- length(tr$tip.label))` `which(tr$edge[, 2] >= n)`

Of course, each of these commands may be wrapped in a function, for instance:

```
getNtips <- function(phy) length(phy$tip.label)
```

4.1 Preorder Tree Traversal

Preorder tree traversal means here: travelling through the tree from the root to the tips. If an object of class `"phylo"` is in cladewise order, then the first element of the first column of `edge` is, by definition, the root and numbered $n + 1$. This is true whether the tree is rooted or not (remind that the root is arbitrary in the latter case). The beginning and end of each clade connected to the root is found with:

```
start <- which(tr$edge[, 1] == length(tr$tip.label) + 1)
end <- c(start[-1] - 1, dim(tr$edge)[1])
```

The MRCA of these clades (i.e., the direct descendants of the root) can be found with:

```
tr$edge[start, 2]
```

As an example, we take the avian families tree:

```
> library(ape)
> data(bird.families)
> n <- length(bird.families$tip.label)
> start <- which(bird.families$edge[, 1] == n + 1)
> end <- c(start[-1] - 1, dim(bird.families$edge)[1])
> start

[1]  1 28

> end

[1]  27 271
```

The two nodes connected to the root are:

```
> bird.families$edge[start, 2]

[1] 139 152
```

This can be checked graphically with:

```
plot(bird.families)
nodelabels()
```

This approach can then be applied repeatedly from the root to the tips. For instance, we find that the first node descendant of the root is `tr$edge[start[1], 2]`: we may thus replace "`n + 1`" above by this value, and "`dim(tr$edge)[1]`" by "`end[1]`":

```
startB <- which(tr$edge[, 1] == tr$edge[start[1], 2])
endB <- c(startB[-1] - 1, end[1])
```

Note the new names `startB` and `endB`; in practice, this may be simplified by using, for instance, recursive calls to a function (see § 4.6).

If the object `tr` is in pruningwise order, then tree traversal may be done through successive search of node and tips numbers using `which` (as was done by most functions in `ape`). However, this is less efficient.

4.2 Finding a Clade

For an arbitrary node, say `nod`, the approach above may be adapted to the following algorithm:

1. Find the node ancestor of `nod`, store its number in `anc`, and store the number of the branch in `i`.
2. Find the next occurrence of `anc` in `tr$edge[, 1]`, store it in `j`.

The clade descending from `nod` is given by the rows $i+1$ to $j-1$ of `tr$edge`. This algorithm coded in R is:

```
i <- which(tr$edge[, 2] == nod)
anc <- tr$edge[i, 1]
tmp <- which(tr$edge[, 1] == anc)
j <- tmp[which(tmp == i) + 1]
tr$edge[(i+1):(j-1), ]
```

Note that it is straightforward to translate this code in C.

4.3 Postorder Tree Traversal

Postorder tree traversal means here: travelling through the tree from the tips to the root. If the object `tr` is in pruningwise order, then postorder tree traversal is straightforward by descending along the rows of `tr$edge`. Otherwise, successive searches must be done.

4.4 Relating Nodes and Tips to Other Data

Use numeric indexing (more efficient) if possible, otherwise use `names` or any other way to assign the value to its node or tip.

4.5 Tracking Nodes

The problem of tracking a node through successive tree manipulation is not easy because nodes are numbered sequentially. For instance, if two trees are binded, the node numbers of one of them must be changed. The solution to this problem is to use the element `node.label`. Node labels may be created easily with:

```
tr$node.label <- paste("node", 1:tr$Nnode, sep = "")
```

If a second tree, say `trb`, is involved here:

```
trb$node.label <- paste("trb_node", 1:trb$Nnode, sep = "")
```

Both trees may be binded now.

```
trc <- bind.tree(tr, trb)
```

To find the node number of say `"trb_node1"` in the new tree:

```
which(trc$node.label == trb_node1)
```

4.6 Example

As an example of using the new definition of the class `"phylo"` and the guidelines above, I rewrote the function `rtree`. The logic of the algorithm is the same in both versions: splitting repeatedly a pool of n tips until splitting cannot be done. But the implementations in R are different. In the old version, the branches were defined successively and the matrix `edge` was filled along its rows; the latter were reordered at the end. In the new version, the rows are filled with respect to the size of the clades using a recursive function, so that no reordering is needed at the end. The following table compares the timings of the two versions.[3]

n	old	new	gain
10	0.026	0.0006	43
100	0.290	0.003	100
1000	4.017	0.030	134

This shows that considerable improvement can be achieved even using R codes only. Furthermore, the algorithm is now more appropriate to be coded in C, so that more improvement seems possible. Note that the operation done by `rtree` is actually a preorder tree traversal: thus, this operation can be done in a very short time with R code.

5 Uniqueness of Representation

At the moment, this remains unsolved bacause even adopting one of the rules described above for the order of the branches may lead to several representations of the same tree. Additional rules are needed. Some open questions:

[3] Timings averaged over 100 consecutive repetitions with a `for` loop, processor at 1.86 GHz, 2 Gb RAM, Linux Knoppix 4.0, R 2.3.0, `ape` 1.8-3.

- Is there an order that makes tree manipulation optimal? (see above for some partial answers)
- Do we really need a standard, unique order?

Keep in mind that the rules should work for rooted and unrooted, dichotomous and multichotomous trees, as well as reticulograms.

6 Passing Trees from R to C

Because the class `"phylo"` uses only vectors,[4] it is easy to pass these data to C using the R function `.C`. For instance, the matrix `edge` may be passed with:

```
.C("nameofCfunction", as.integer(tr$edge),
   as.integer(dim(tr$edge)[1]), PACKAGE = "nameofpackage")
```

which will be received in C with:

```
void nameofCfunction(int * edge, int * n)
```

where `edge` is a pointer to an array of size $2n$. The two nodes of the ith branch will be accessed with `edge[i - 1]` and `edge[i - 1 + n]`.

An alternative, maybe easier, solution is to pass separately the two columns of `edge`:

```
.C("nameofCfunction", as.integer(tr$edge[, 1]),
   as.integer(tr$edge[, 2]),
   as.integer(dim(tr$edge)[1]), PACKAGE = "nameofpackage")
```

with in the C program:

```
void nameofCfunction(int * edge1, int * edge2, int * n)
```

Now the two nodes will be accessed with `edge1[i - 1]` and `edge2[i - 1]`. A complete tree structure may be passed with `.C`:

```
.C("nameofCfunction", as.integer(tr$edge), as.integer(tr$Nnode),
   as.integer(dim(tr$edge)[1]), as.double(tr$edge.length),
   as.character(tr$tip.label), as.character(tr$node.label),
   PACKAGE = "nameofpackage")
```

received in C with:

```
void nameofCfunction(int *edge, int *nnode,  int *n,
                     double *edge_length, char **tip_label,
                     char **node_label)
```

Note that other elements may be added in the arguments as long as they are R vectors. The advantage of using `.C` is that data manipulation in C is similar to any program in this language: the only constraint is that the function receiving the data from R must have pointers and return void.

Using `.Call` or `.External` is more flexible on the R side:

[4] Matrices in R are actually vectors.

```
.Call("nameofCfunction", tr, PACKAGE = "nameofpackage")
.External("nameofCfunction", tr, PACKAGE = "nameofpackage")
```

where `tr` may be any R data. But the data manipulation in C is more complex since it deals with SEXP (*S expression*) structures:

```
SEXP nameofCfunction(SEXP tr)
```

This is advantageous if the number and/or size of elements is unknown in advance. An example can be found in the sources of `ape` (see src/bipartition.c and R/dist.topo.R).

7 Others

It is also possible to add further attributes to an object of class `"phylo"` such as the number of tips (n), ... also `rooted` (TRUE or FALSE), `order` (`"cladewise"` or `"pruningwise"`), and so on. This may use `attr` or `$`.

In fact, any element may be appended in an object of class `"phylo"` since it is a list. Adding some information may be useful to store the results of previous computations (as done with the class `"phylog"` in `ade4`).

8 Future Developments

The development of `ape` goes along many lines. Three points need to be mentioned in the present context:

- Finishing a completely documented API;
- Breaking `ape` in several (five?) packages;
- Developing an interface between the coding described here and the traditional one [2] at the C-level.

9 References

[1] Diaconis P. W. & Holmes S. P. 1998. Matchings and phylogenetic trees. *Proceedings of the National Academy of Sciences USA* **95**: 14600–14602.

[2] Felsenstein J. 2004. *Inferring phylogenies*. Sinauer Associates, Sunderland, Mass., USA.

[3] Paradis E. 2006. *Analysis of phylogenetics and evolution with R*. Springer, New York.

A Memory Requirements (bis)

The following figures were obtained with R 2.4.0.

tree	n	old	new	[a]	gain	[a]
bird.orders	23	5112	2712	(2360)	1.88	(2.17)
bird.families	135	28,280	13,168	(11,000)	2.15	(2.57)
chiroptera	917	150,264	75,080	(64,328)	2.00	(2.34)
random tree	1000	196,368	84,544	(68,560)	2.32	(2.86)
"	10,000	1,960,368	840,544	(680,560)	2.33	(2.88)

[a] forcing storage mode as integer when possible

The figures in parentheses are when the storage mode of integer values is forced to be integer, e.g.:

```
> library(ape)
> tr <- rtree(1000)
> object.size(tr)

[1] 84544

> storage.mode(tr$edge)

[1] "double"

> storage.mode(tr$edge) <- "integer"
> storage.mode(tr$Nnode) <- "integer"
> object.size(tr)

[1] 68560
```