Using **ggphylo** and **ggplot** to visualize phylogenetic trees and alignments

Gregory Jordan

May 3, 2012

ggphylo is a package that provides convenient functions for manipulating `phylo` objects from R and plotting them using **ggplot**. External data can be attached to trees via NHX-format files, data frames or CSV files. These data can be mapped to visual elements of the tree or trees, allowing one to construct complex visualizations in a simple, flexible way.

1 For the impatient

1.1 Basic plots

To get started, simply input a `phylo` object (or a list of `phylo` objects) and call the `ggphylo` function:

```
> tree.list <- list()
> for (i in 1:3) {
   x <- rtree(20)  # Random trees 20 leaves.
   tree.list[[i]] <- x
 }
> ggphylo(tree.list) # Plot the list of tres.
```
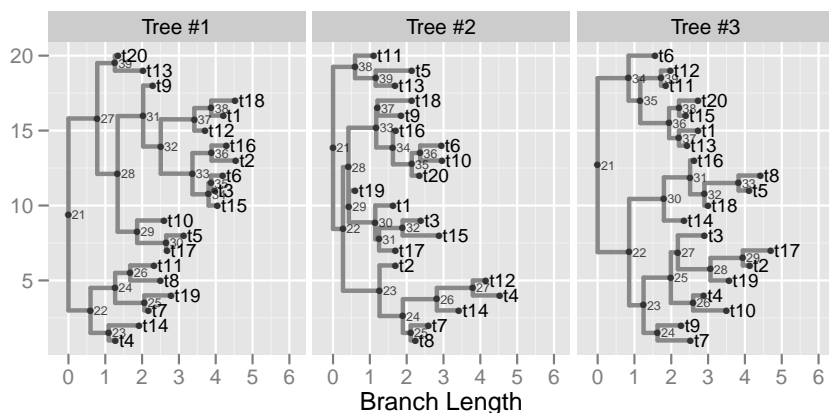

The result looks similar to the standard `plot.phylo` function from the `ape` package. It also scales well to several trees:

```
> n <- 12
> sizes <- sample(2:20, n, replace=T)
> for (i in 1:n) {
   tree.list[[i]] <- rtree(sizes[i])
 }
> ggphylo(tree.list, label.size=2) # Plot the list of trees.
```
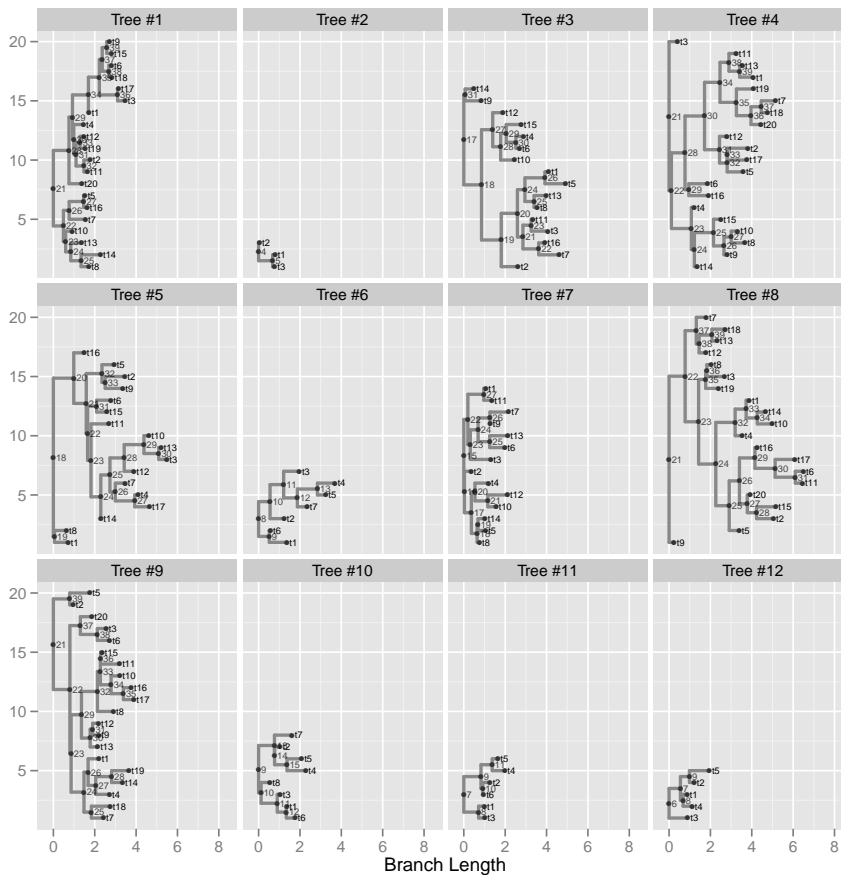
1.2 Plotting data along trees

The ggphylo package defines four visual entities (**lines, nodes, labels, and internal.labels**) and three visual properties (**color, alpha, size**) for visualizing data along a tree. Any combination of entity and property can be used: for example, the following code maps **bootstrap** values to **line color**, and **population size** values to **node size**:

```
> n <- 40;  x <- rtree(n); n.nodes <- length(nodes(x))
> bootstraps <- 100 - rexp(n.nodes, rate=5) * 100
> pop.sizes <- pmax(0, rnorm(n.nodes, mean=50000, sd=50000))
> for (i in nodes(x)) {
   x <- tree.set.tag(x, i, 'bootstrap', bootstraps[i])
   x <- tree.set.tag(x, i, 'pop.size', pop.sizes[i])
 }
> plot.args <- list(
   x,
   line.color.by='bootstrap',
   line.color.scale=scale_colour_gradient(limits=c(50, 100), low='red', high='black'),
   node.size.by='pop.size',
   node.size.scale = scale_size_continuous(limits=c(0, 100000), range=c(1, 5)),
   label.size=2
 )
> do.call(ggphylo, plot.args)
```
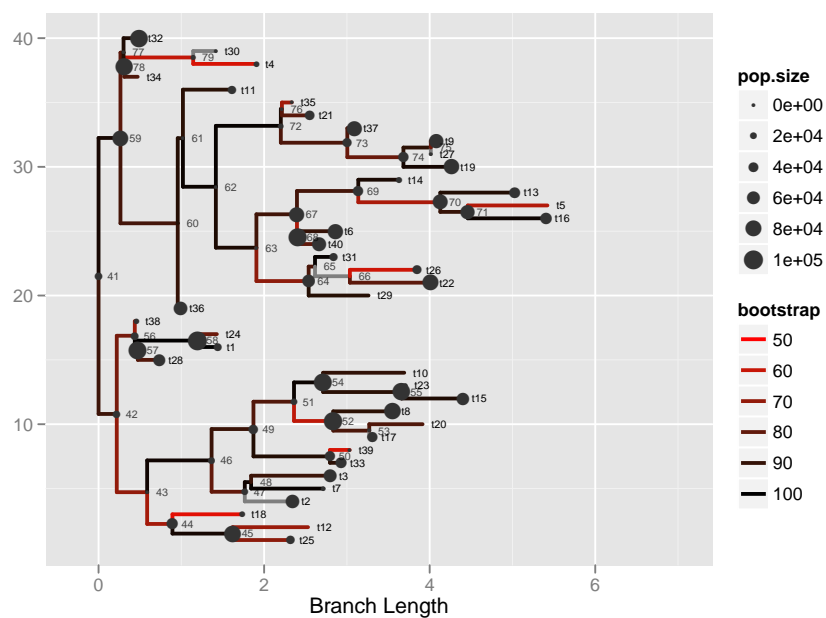
```
> unrooted.args <- plot.args
> unrooted.args[['layout']] <- 'unrooted'
> do.call(ggphylo, unrooted.args)
```
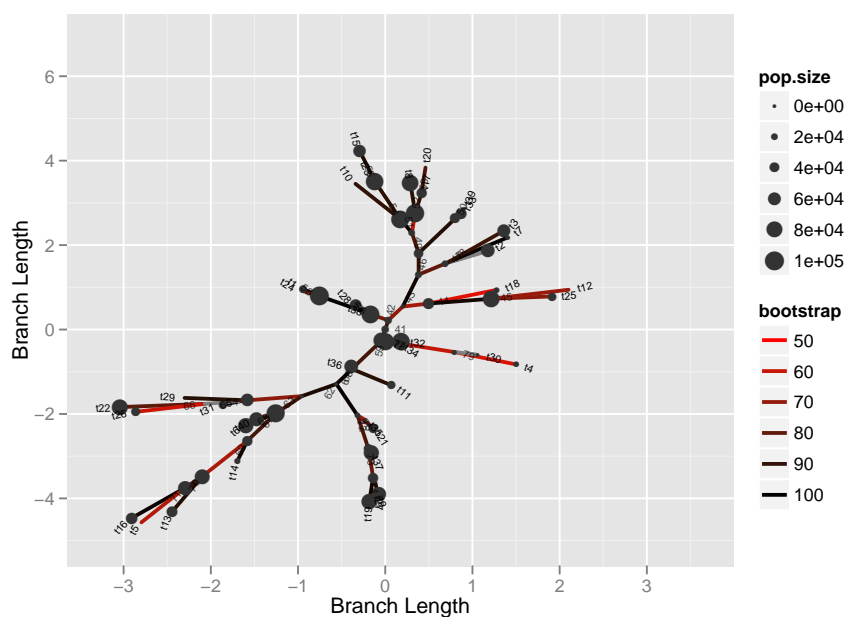


```
> radial.args <- plot.args
> radial.args[['layout']] <- 'radial'
> do.call(ggphylo, radial.args)
```
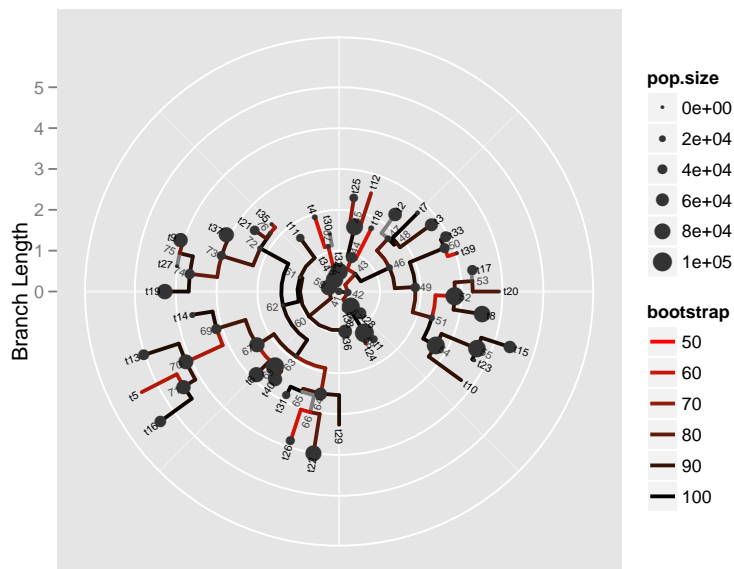
Details of the above code will be explained in the remaining sections, but the general approach should be somewhat familiar to existing ggplot users. This introductory section was designed to merely give a brief overview of the main functionality exposed by ggphylo.

2 Using ggphylo

2.1 Attaching annotations to trees

To visualize data along a phylogeny, it must first be associated with nodes in the tree. ggphylo uses "tags" to store a set of key-value pairs with each node. (Internally, this is done by creating a list of lists, one per node, attached to the phylo object.) Data can be associated with nodes of a tree in three ways. Let's explore those methods by working with simple tree objects and attaching data in each of three ways:

- `tree.set.tag` This function allows a single key-value pair to be attached to a given node in the tree. The `phylo` object, node index, key and value are required arguments:

```
> tree <- tree.read('((a,b),c);')
> # Use the 'tree.find' method to return the node index given a label.
> b.node.index <- tree.find(tree, 'b')

[1] 2

> print(b.node.index)

[1] 2

> x <- tree.set.tag(tree, tree.find(tree, 'b'), 'foo', 'bar')

[1] 2

> print(as.character(x))

[1] "((a,b[&&NHX:foo=bar]),c);"

> print(as.data.frame(x, minimal.columns=T))
```

```
  label node  foo
1     a    1 <NA>
5  <NA>    5 <NA>
2     b    2  bar
4  <NA>    4 <NA>
3     c    3 <NA>
```

Note that the above code uses a few of the convenience functions exposed by `ggphylo`: `tree.find` for finding the node index which has a given label, `as.character` which converts a tree into a Newick or NHX formatted string, and `as.data.frame` which converts a `phylo` object into a `data.frame` with one row per node. The `minimal.columns=T` option causes only the bare minimum information about each node to be included; a more detailed data frame can be produced by setting `minimal.columns=F`;

- `tree.read.nhx` The NHX format allows key-value pairs to be stored within a Newick-formatted tree string. `ggphylo` contains a tree.read.nhx function which extracts this data from a NHX string and stores it as tags in the returned `phylo` object:

```
> x <- tree.read.nhx('((a,b[&&NHX:foo=bar]),c[&&NHX:bizz=buzz]);')
[1] 2
[1] 3
> print(as.data.frame(x, minimal.columns=T))
  label node  foo bizz
1     a    1 <NA> <NA>
5  <NA>    5 <NA> <NA>
2     b    2  bar <NA>
4  <NA>    4 <NA> <NA>
3     c    3 <NA> buzz
```

It's useful to test how robust the tree parsing and writing functions are by performing round-trip tests:

```
> str <- '(((a[&&NHX:x=1],b[&&NHX:x=2])c[&&NHX:y=3],d[&&NHX:y=4])e,f)g;'
> x <- tree.read.nhx(str)
[1] 1
[1] 2
[1] 7
[1] 3
> print(as.character(x) == str)
[1] TRUE
```

- `tree.load.data` Sometimes the NHX format can be cumbersome for storing data when it is more easily available in row-based formats. In this case, a `data.frame` or CSV file can be used to add tags to a tree using the `tree.load.data` function. The only requirement is a 'label' column in the source data, which is used to match unique labels in the data and in the tree:

```
> tree <- tree.read('((a,b),c);')
> x <- data.frame(
    label=c('a', 'b', 'c'),
    xyz=c(1, 2, 3)
  )
> print(x)
```

```
  label xyz
1     a   1
2     b   2
3     c   3

> tree <- tree.load.data(tree, x)

[1] 1
[1] 2
[1] 3

> print(as.data.frame(tree, minimal.columns=T))

  label node xyz
1     a    1   1
5  <NA>    5  NA
2     b    2   2
4  <NA>    4  NA
3     c    3   3
```

2.2   Controlling visual mappings