Visualizing and Comparing Trees of Heirarchical Clustering: Using the dendextend R Package

Tal Galili
Tel-Aviv University

Yoav Benjamini
Tel-Aviv University

Abstract

A dendrogram is a tree diagram which is often used to visualize an hierarchical clustering of items. Dendrograms are used in many dispelines, ranging from Phylogenetic Trees in computational biology to Lexomic Trees in text analysis.

The **dendextend** package extends the dendrogram objects in the R programming language, allowing for easy manipulation of a dendrogram's shape, color and content. Furthermore, it enable the tools for comparing the similarity of two dendrograms to one another both graphically (using tanglegrems) and statistically (from cophenetic correlations to Bk plots).

The paper gives a detailed exposition of both the internal structure of the package and the provided user interfaces.

Keywords: Dendrogram, hclust, hierarchical clustering, visualization, tanglegram, R.

1. Introduction

1.1. The `dendrogram` object

The `dendrogram` class provides general functions for handling tree-like structures in R (R Development Core Team 2013). It is intended as a replacement for similar functions in hierarchical clustering and classification/regression trees, such that all of these can use the same engine for plotting or cutting trees.

A dendrogram object represents a tree as a nested `list` object, with various attributes.

Dendrogram has several useful methods bundled with R:

```
methods(class = "dendrogram")

##  [1] [[.dendrogram*           as.hclust.dendrogram*
##  [3] as.phylo.dendrogram      cophenetic.dendrogram*
##  [5] cut.dendrogram*          cutree.dendrogram
##  [7] entanglement.dendrogram* head.dendrogram*
##  [9] labels.dendrogram*       labels<-.dendrogram
## [11] merge.dendrogram*        nleaves.dendrogram*
## [13] nnodes.dendrogram*       plot.dendrogram*
```

```
## [15] print.dendrogram*        reorder.dendrogram*
## [17] rev.dendrogram*          rotate.dendrogram*
## [19] sort.dendrogram*         str.dendrogram*
## [21] tanglegram.dendrogram*   trim.dendrogram*
## [23] unbranch.dendrogram*
##
##    Non-visible functions are asterisked
```

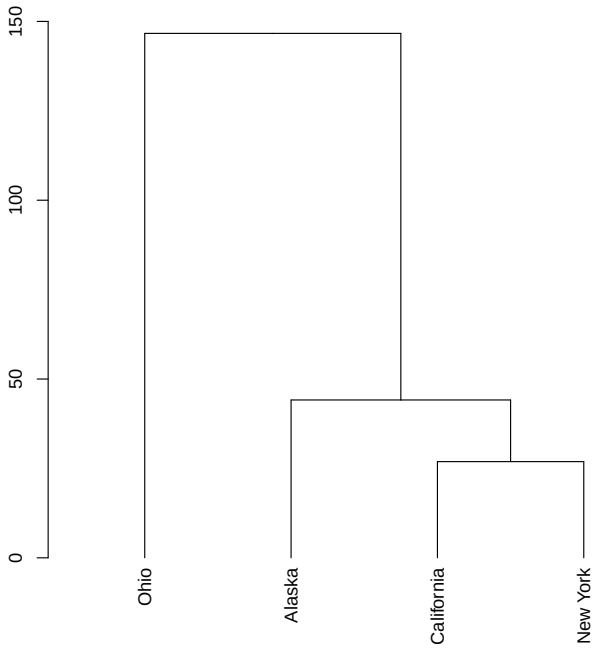
For example, let's create a dendrogram object based on an hierarchical clustering of 4 states in the U.S.:

```
# our data:
data(USArrests)
US_data <- USArrests[c(2, 5, 32, 35), ]
print(US_data)

##            Murder Assault UrbanPop Rape
## Alaska       10.0     263       48 44.5
## California    9.0     276       91 40.6
## New York     11.1     254       86 26.1
## Ohio          7.3     120       75 21.4

hc <- hclust(dist(US_data), "ave")  # create an heirarchical clustering object
dend <- as.dendrogram(hc)
```

Below are examples for some dendrogram methods:

```
print(dend)

## 'dendrogram' with 2 branches and 4 members total, at height 146.7

labels(dend)

## [1] "Ohio"       "Alaska"     "California" "New York"

str(dend)

## --[dendrogram w/ 2 branches and 4 members at h = 147]
##   |--leaf "Ohio"
##   `--[dendrogram w/ 2 branches and 3 members at h = 44.1]
##      |--leaf "Alaska"
##      `--[dendrogram w/ 2 branches and 2 members at h = 26.9]
##         |--leaf "California"
##         `--leaf "New York"

str(dend[[2]])  # looking at one branch of the dendrogram
```

```
## --[dendrogram w/ 2 branches and 3 members at h = 44.1]
##   |--leaf "Alaska"
##   `--[dendrogram w/ 2 branches and 2 members at h = 26.9]
##      |--leaf "California"
##      `--leaf "New York"

plot(dend)
```



You might notice how the order of the items (leaves/terminal nodes) of the dendrogram is different than their order in the table. In order to re-order the rows in the data-table to have the same order as the items in the dendrogram, we can use the `order.dendrogram` function:

```
(new_order <- order.dendrogram(dend))

## [1] 4 1 2 3

# the order of the original items to have them be at the same
# order as they assume in the dendrogram
print(US_data[new_order, ])

##            Murder Assault UrbanPop Rape
## Ohio          7.3     120       75 21.4
## Alaska       10.0     263       48 44.5
## California    9.0     276       91 40.6
## New York     11.1     254       86 26.1
```

In order to see what our dendrogram (`list`) object includes, we need to use the `unclass` function, which will strip away the class attribute and will allow us to print the list as is, without going through the `print.dendrogram` method. We can see how each node in the dendrogram/list object has the following (self explaining) attributes:

```
str(unclass(dend))

## List of 2
##  $ : atomic [1:1] 4
##   ..- attr(*, "members")= int 1
##   ..- attr(*, "height")= num 0
##   ..- attr(*, "label")= chr "Ohio"
##   ..- attr(*, "leaf")= logi TRUE
##  $ :List of 2
##   ..$ : atomic [1:1] 1
##   .. ..- attr(*, "members")= int 1
##   .. ..- attr(*, "height")= num 0
##   .. ..- attr(*, "label")= chr "Alaska"
##   .. ..- attr(*, "leaf")= logi TRUE
##   ..$ :List of 2
##   .. ..$ : atomic [1:1] 2
##   .. .. ..- attr(*, "label")= chr "California"
##   .. .. ..- attr(*, "members")= int 1
##   .. .. ..- attr(*, "height")= num 0
##   .. .. ..- attr(*, "leaf")= logi TRUE
##   .. ..$ : atomic [1:1] 3
##   .. .. ..- attr(*, "label")= chr "New York"
##   .. .. ..- attr(*, "members")= int 1
##   .. .. ..- attr(*, "height")= num 0
##   .. .. ..- attr(*, "leaf")= logi TRUE
##   .. ..- attr(*, "members")= int 2
##   .. ..- attr(*, "midpoint")= num 0.5
##   .. ..- attr(*, "height")= num 26.9
##   ..- attr(*, "members")= int 3
##   ..- attr(*, "midpoint")= num 0.75
##   ..- attr(*, "height")= num 44.1
##  - attr(*, "members")= int 4
##  - attr(*, "midpoint")= num 0.875
##  - attr(*, "height")= num 147
```

Notice how terminal nodes uses the ”leaf” attribute (set to TRUE).

```
names(attributes(dend)[-4])

## [1] "members"  "midpoint" "height"
```

A very important function is `dendrapply`. It applies some function recursively to each node of a dendrogram. It is often used for adjusting attributes of the object, or extracting something from it.

One current "feature" with this function is that just sending a dendrogram through it will return it with each of its nodes becoming of class "dendrogram". Notice the use of the `unclass_dend` function. Example:

```
# dendrapply(dend, unclass) # in case the
itself <- function(x) x
dend_from_dendrapply <- dendrapply(dend, itself)

# here we must first use unclass since '[[]]' inherits its
# class to the output:
class(unclass(dend)[[2]])

## [1] "list"

class(unclass(dend_from_dendrapply)[[2]])

## [1] "dendrogram"

class(unclass_dend(dend_from_dendrapply)[[2]])  # the new uncless_dend solves it.

## [1] "list"
```

1.2. Motivation for creating `dendextend`

The `dendrogram` object has several **advantages**:

1. `dendrogram` objects are list R objects. This makes their structure very familiar and easy to understand by R users. They are also, relatively, simple to manipulate and extend.
2. `dendrogram` objects has various methods and functions for using them within R base.
3. Other tree objects, such as `hclust`, and objects from the **ape** package (Paradis *et al.* 2004), include an `as.dendrogram` method for converting their objects into a dendrogram. And also `as.phylo.dendrogram`, `as.hclust.dendrogram`.
4. `dendrogram` objects are used in various packages as an intermediate step for other purposes (often plotting), such as:
   (a) The **latticeExtra** package (Sarkar and Andrews 2012), see the `dendrogramGrob` function.
   (b) The **labeltodendro** package (Nia and Stephens 2011), see the `colorplot` function.
   (c) The **bclust** package (Nia and Davison 2012), see the `bclust` function.
   (d) The **ggdendro** package (de Vries and Ripley 2013), see the `dendro_data` function.

(e) The **Heatplus** package (Ploner 2012), see the `annHeatmap2` function.

(f) The **sparcl** package (Ploner 2012), see the `ColorDendrogram` function.

However, even with all of its advantages, the `dendrogram` class in R still lacks various basic features.

The `dendextend` package aims at filling some gaps in base R, by extending the available functions for dendrogram manipulation, statistical analysis, and visualization.

This vignettes Provides a step-by-step description of the functionality provided by the `dendextend` package.

1.3. Installing `dendextend`

To install the stable version from CRAN use:

```
install.packages("dendextend")  # not yet available from CRAN
```

To install the GitHub version use:

```
if (!require("devtools")) install.packages("devtools")
require("devtools")
install_github("dendextend", "talgalili")
```

2. Tree attributes (extraction, assignment, length)

2.1. labels in base R

In base R, the `labels` function is intended to find/extract a suitable set of labels from an object for use in printing or plotting, for example. By default, it uses the `names` and `dimnames` functions.

What base R `labels` function is missing is assignment. In the next few examples we will go through different examples of what the `dendextend` package offers for various objects.

Credits: These assignment functions were originally written by Gavin Simpson (in a post on (stackoverflow)), and adopted/adjusted to this package by Tal Galili. Some modification were inspired by Gregory Jefferis's code from the **dendroextras** package.

2.2. labels for vectors and matrices

In base R, for vectors, labels gives the `names` of the object. And if these are missing, then `labels` will give the vector itself as a character vector:

```
x <- 1:3
names(x)  # this vector has no names

## NULL
```

```
labels(x)  # this vector has no labels

## [1] "1" "2" "3"
```

Assignment to names is available in base R and works as follows:

```
x <- 1:3
names(x) <- letters[1:3]  # assignment for names is in base R
# both names and labels will give the same result:
names(x)

## [1] "a" "b" "c"

labels(x)

## [1] "a" "b" "c"
```

The new labels assignment function will allow a user to change the labels of the vector just as if it was "names":

```
x <- 1:3
labels(x) <- letters[1:3]
names(x)

## [1] "a" "b" "c"

labels(x)

## [1] "a" "b" "c"
```

Labels assignment are also available for matrices.

2.3. labels for dendrogram objects

We can get a dendrogram's labels using the `labels` function from base R. However, in order to assign new values to it, we'll need the assignment function from **dendextend**:

```
labels(dend)  # from base R

## [1] "Ohio"       "Alaska"     "California" "New York"

set.seed(2354235)
labels(dend) <- sample(labels(dend))  # labels assingment - thanks to dendextend
labels(dend)

## [1] "New York"   "Ohio"       "Alaska"     "California"
```

2.4. labels for hclust objects

dendextend offers a `labels` method for `hclust` objects. It take special care to have the order of the labels be the same as is with dendrogram object, which is the order of the labels in the plotted tree. This can be turned off when using the `order` parameter:

```
# All are from dendextend
labels(hc)

## [1] "Ohio"       "Alaska"     "California" "New York"

labels(hc, order = FALSE)  # this is the order of the rows of the original data.

## [1] "Alaska"     "California" "New York"   "Ohio"

set.seed(229835)
labels(hc) <- sample(labels(hc))  # labels assingment - thanks to dendextend
labels(hc)

## [1] "California" "New York"   "Alaska"     "Ohio"
```

2.5. labels assignment and recycling

When the assigned vector has a different length, the **dendextend** assignment functions will recycle the value but also give a warning:

```
x <- 1:3
hc <- hclust(dist(US_data), "ave")
dend <- as.dendrogram(hc)
y <- matrix(1:9, 3, 3)

labels(x) <- "bob"

## Warning:  The lengths of the new labels is shorter than the length of the object
- labels are recycled.

labels(x)

## [1] "bob" "bob" "bob"

labels(hc) <- "bob"

## Warning:  The lengths of the new labels is shorter than the number of leaves
in the hclust - labels are recycled.

labels(hc)
```

```
## [1] "bob" "bob" "bob" "bob"

labels(dend) <- "bob"

## Warning:  The lengths of the new labels is shorter than the number of leaves
in the dendrogram - labels are recycled.

labels(dend)

## [1] "bob" "bob" "bob" "bob"

labels(y) <- "bob"

## Warning:  The lengths of the new labels is shorter than the length of the object's
colnames - labels are recycled.

labels(y)

## [1] "bob" "bob" "bob"
```

2.6. Tree size - number of leaves

Getting the size of a tree (e.g: number of leaves/terminal-nodes) is good for validation of functions, and also when we wish to initiate a variable to later fill with data from the leaves.

The `labels` function for dendrogram is expensive, since it uses recursion to get all of the tree's elements. If we are only interested in getting the tree size, it is better to use the `nleaves` function. It has an S3 method for hclust, dendrogram and phylo (from the **ape**):

```
nleaves(hc)

## [1] 4

nleaves(dend)

## [1] 4
```

For dendrograms the speed improvement is about 10 times using `labels`, whereas for hclust, there is not any gain made by using `nleaves`. Here is a quick benchmark:

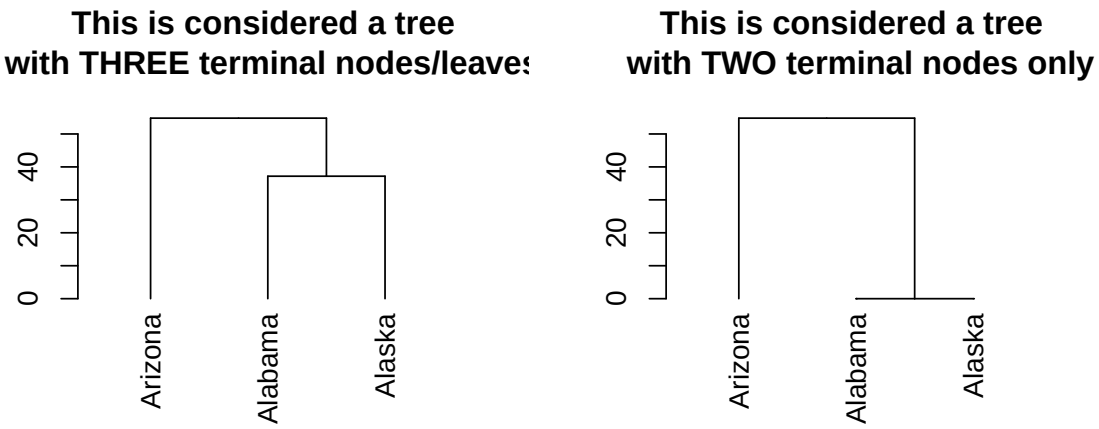
```
library(microbenchmark)
microbenchmark(nleaves(dend), length(labels(dend)))

## Unit: microseconds
##                 expr    min     lq median    uq   max
```

```
##         nleaves(dend)  24.64  25.76  26.88  30.8 212.8
##  length(labels(dend)) 390.83 396.99 400.35 406.5 951.3
##  neval
##    100
##    100

microbenchmark(nleaves(hc), length(labels(hc)))

## Unit: microseconds
##                expr   min    lq median    uq    max neval
##         nleaves(hc) 19.60 20.16  20.72 21.28  31.92   100
##  length(labels(hc)) 31.36 31.92  32.20 32.48 151.18   100
```

There are border-line cases where the node above some leaves is of height 0. In such a case, we would consider that node as a "terminal node", and in order to count the number of such terminal nodes we would use `count_terminal_nodes` function. For example:

```
hc <- hclust(dist(USArrests[1:3, ]), "ave")
dend <- as.dendrogram(hc)

par(mfrow = c(1, 2))

### Trivial case
count_terminal_nodes(dend)  # 3 terminal nodes

## [1] 3

length(labels(dend))  # 3 - the same number

## [1] 3

plot(dend, main = "This is considered a tree \n with THREE terminal nodes/leaves")

### NON-Trivial case
str(dend)

## --[dendrogram w/ 2 branches and 3 members at h = 54.8]
##   |--leaf "Arizona"
##   `--[dendrogram w/ 2 branches and 2 members at h = 37.2]
##      |--leaf "Alabama"
##      `--leaf "Alaska"

attr(dend[[2]], "height") <- 0
count_terminal_nodes(dend)  # 2 terminal nodes, why? see this plot:

## [1] 2
```

```
# while we have 3 leaves, in practice we have only 2 terminal
# nodes (this is a feature, not a bug.)
plot(dend, main = "This is considered a tree \n with TWO terminal nodes only")
```



2.7. Tree size - number of nodes

Getting the size of a tree, in terms of the number of nodes can easily be done using:

```
hc <- hclust(dist(USArrests[1:3, ]), "ave")
dend <- as.dendrogram(hc)

nnodes(hc)

## [1] 5

nnodes(dend)

## [1] 5
```

2.8. Generally getting tree attributes

Getting tree attributes can more generally be achieved using `get_nodes_attr`, however, the dedicated function are often faster than the general solution. (also, in the future, we might introduce functions based on **Rcpp**, offering even faster times).

Here are some examples:

```
hc <- hclust(dist(USArrests[1:3, ]), "ave")
dend <- as.dendrogram(hc)

# get_leaves_attr(dend) # error :)
get_leaves_attr(dend, "label")
```

```
## [1] "Arizona" "Alabama" "Alaska"

labels(dend, "label")

## [1] "Arizona" "Alabama" "Alaska"

get_leaves_attr(dend, "height")  # should be 0's

## [1] 0 0 0

get_nodes_attr(dend, "height")

## [1] 54.80  0.00 37.18  0.00  0.00

get_branches_heights(dend, sort = FALSE)  # notice the sort=FALSE

## [1] 54.80 37.18

get_leaves_attr(dend, "leaf")  # should be TRUE's

## [1] TRUE TRUE TRUE

get_nodes_attr(dend, "leaf")  # conatins NA's

## [1]   NA TRUE   NA TRUE TRUE

get_nodes_attr(dend, "leaf", na.rm = TRUE)  #

## [1] TRUE TRUE TRUE

get_leaves_attr(dend, "members")  # should be 1's

## [1] 1 1 1

get_nodes_attr(dend, "members", include_branches = FALSE, na.rm = TRUE)  #

## [1] 1 1 1

get_nodes_attr(dend, "members")  #

## [1] 3 1 2 1 1

get_nodes_attr(dend, "members", include_leaves = FALSE, na.rm = TRUE)  #

## [1] 3 2
```

```
hang_dend <- hang.dendrogram(dend)
get_leaves_attr(hang_dend, "height")  # no longer 0!

## [1] 49.32 31.70 31.70

get_nodes_attr(hang_dend, "height")  # does not include any 0s!

## [1] 54.80 49.32 37.18 31.70 31.70

# does not include leaves values:
get_nodes_attr(hang_dend, "height", include_leaves = FALSE)

## [1] 54.80    NA 37.18    NA    NA

# remove leaves values all together:
get_nodes_attr(hang_dend, "height", include_leaves = FALSE, na.rm = TRUE)

## [1] 54.80 37.18

get_branches_heights(hang_dend)  # notice the sort

## [1] 37.18 54.80

get_branches_heights(hang_dend, sort = FALSE)  # notice the sort

## [1] 54.80 37.18
```

Quick comparison on fetching leaves attributes:

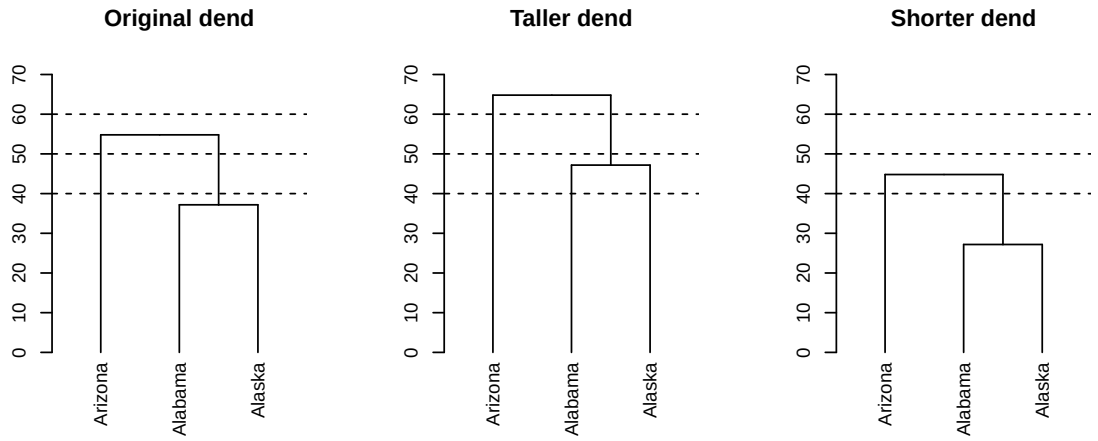
```
require(microbenchmark)
# get_leaves_attr is twice faster than get_nodes_attr
microbenchmark(get_leaves_attr_4members = get_leaves_attr(dend,
    "members"), get_nodes_attr_4members = get_nodes_attr(dend,
    "members", include_branches = FALSE, na.rm = TRUE))

## Unit: microseconds
##                       expr   min    lq median    uq    max
##  get_leaves_attr_4members 290.6 295.1  300.7 324.2  444.6
##   get_nodes_attr_4members 708.9 717.3  732.4 786.1 2855.6
##  neval
##    100
##    100
```

3. Tree manipulation

3.1. unbranching and root height

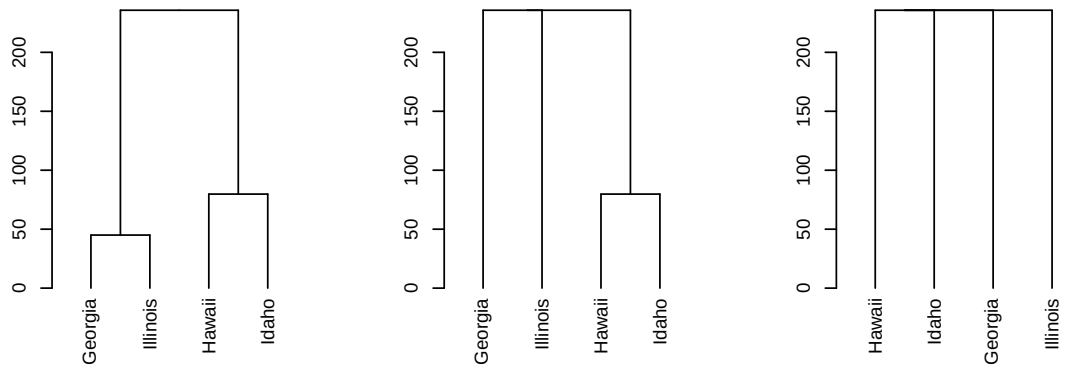
A tree's nodes has various heights. Sometimes we are interested in changing the height of the entire tree. It is useful when This can be accomplished using `raise.dendrogram`. For example (notice how the entire tree's height is changed):

```
hc <- hclust(dist(USArrests[1:3, ]), "ave")
dend <- as.dendrogram(hc)

taller_dend <- raise.dendrogram(dend, 10)
shorter_dend <- raise.dendrogram(dend, -10)

attr(dend, "height")  # 54.80041

## [1] 54.8

attr(taller_dend, "height")  # 64.80041

## [1] 64.8

attr(shorter_dend, "height")  # 44.80041

## [1] 44.8

par(mfrow = c(1, 3))
plot(dend, ylim = c(0, 70), main = "Original dend")
abline(h = c(40, 50, 60), lty = 2)
plot(taller_dend, ylim = c(0, 70), main = "Taller dend")
abline(h = c(40, 50, 60), lty = 2)
plot(shorter_dend, ylim = c(0, 70), main = "Shorter dend")
abline(h = c(40, 50, 60), lty = 2)
```



Sometimes we wish to "unbranch" the dendrogram, meaning that we merge one of the tree's branches with its root. This is useful, for example, when merging phylogenetic trees from several families, and being unwilling to assume a specific root/height to the merged trees. unbranching can be done using the `unbranch` (S3) function (notice the use of the `branch_becoming_root` parameter):

```
hc <- hclust(dist(USArrests[10:13, ]), "ward")
dend <- as.dendrogram(hc)

unbranched_dend <- unbranch(dend, branch_becoming_root = 1)
unbranched_dend_2 <- unbranch(unbranched_dend, branch_becoming_root = 3)

par(mfrow = c(1, 3))
plot(dend)
plot(unbranched_dend)
plot(unbranched_dend_2)
```

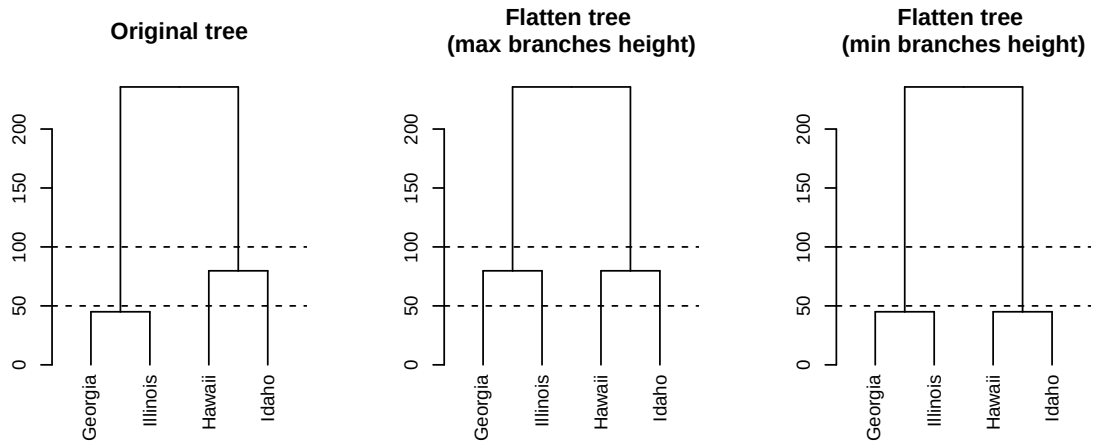


While the `unbranch.hclust` method exists, it is not expected to work since `hclust` objects are

not designed to handle non-binary trees (hence the advantage of using `dendrogram` objects). For `phylo` objects (from the **ape** package), there is also a method that would simply use `ape:::unbranch(phy = x)`.

In some rare cases, we might wish to equalize the heights of root's branches. For this we can use the `flatten.dendrogram` function:

```
hc <- hclust(dist(USArrests[10:13, ]), "ward")
dend <- as.dendrogram(hc)

flatten_dend_1 <- flatten.dendrogram(dend, FUN = max)
flatten_dend_2 <- flatten.dendrogram(dend, FUN = min)

par(mfrow = c(1, 3))
plot(dend, main = "Original tree")
abline(h = c(50, 100), lty = 2)
plot(flatten_dend_1, main = "Flatten tree \n(max branches height)")
abline(h = c(50, 100), lty = 2)
plot(flatten_dend_2, main = "Flatten tree \n(min branches height)")
abline(h = c(50, 100), lty = 2)
```

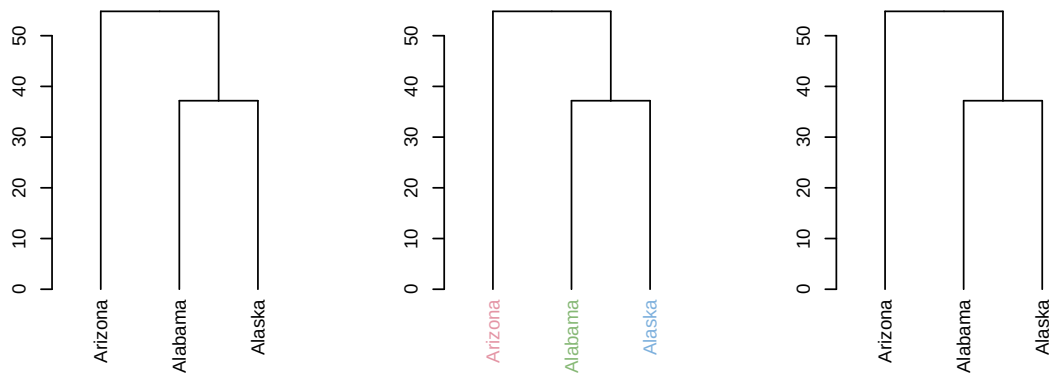


3.2. Coloring labels of leaves

Coloring labels can sometimes be useful, it is done through the `labels_colors` function (which also has assignemnt). Notice the assignment recycling, as well as the differene in the appearence of a dot when labels' color is black, compared to when it is NULL:

```
par(mfrow = c(1, 3))

hc <- hclust(dist(USArrests[1:3, ]), "ave")
dend <- as.dendrogram(hc)
```

```
# Defaults:
labels_colors(dend)

## NULL

plot(dend)

# let's add some color:
require(colorspace)
labels_colors(dend) <- rainbow_hcl(3)
labels_colors(dend)

##   Arizona   Alabama    Alaska
## "#E495A5" "#86B875" "#7DB0DD"

plot(dend)

# changing color to black
labels_colors(dend) <- 1

## Warning:  Length of color vector was shorter than the number of leaves - vector
color recycled

labels_colors(dend)

## Arizona Alabama  Alaska
##       1       1       1

plot(dend)
```

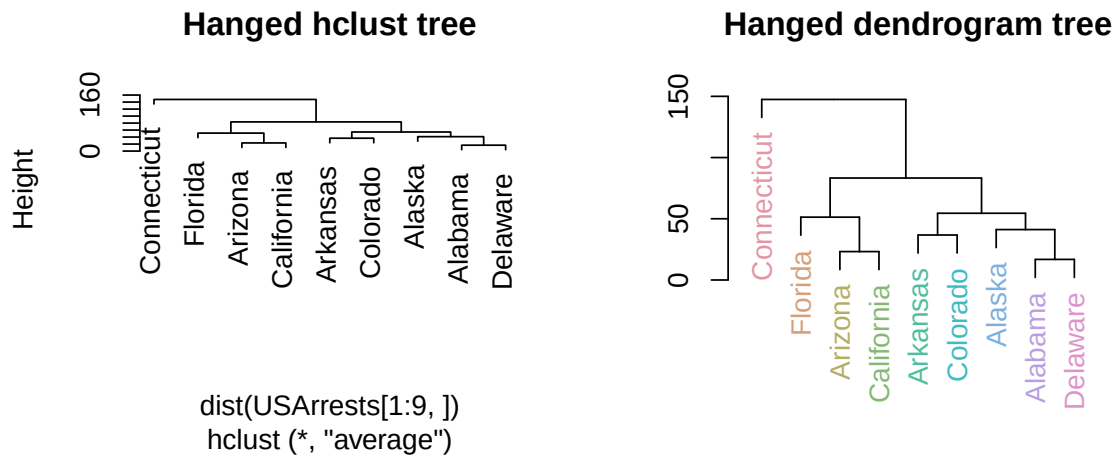


```
# removing color (and the nodePar completely - if it has no
# other attributed but lab.col)
labels_colors(dend) <- NULL

## Warning:  Length of color vector was shorter than the number of leaves - vector
color recycled
## Warning:  'x' is NULL so the result will be NULL

labels_colors(dend)

## NULL
```

3.3. Hanging a dendrogram

Hanging a tree means that we change the height of the leaves to be near their parent node. Hanging helps when examining the toplogy of the tree. Currently, hanging of a dendrogram was possible by going through the hclust object, but now you can simply use the `hang.dendrogram` function. Here is an example:

```
hc <- hclust(dist(USArrests[1:9, ]), "ave")
dend <- as.dendrogram(hc)
hang_dend_1 <- hang.dendrogram(dend, hang = 0.1)
hang_dend_2 <- as.dendrogram(hc, hang = 0.1)  # another way of doing it, if
# we could move from/to hclust
identical(hang_dend_1, hang_dend_2)  # and they are the same :)

## [1] TRUE

require(colorspace)
labels_colors(hang_dend_1) <- rainbow_hcl(nleaves(hang_dend_1))

par(mfrow = c(1, 2))
plot(hc, main = "Hanged hclust tree")
plot(hang_dend_1, main = "Hanged dendrogram tree")
```

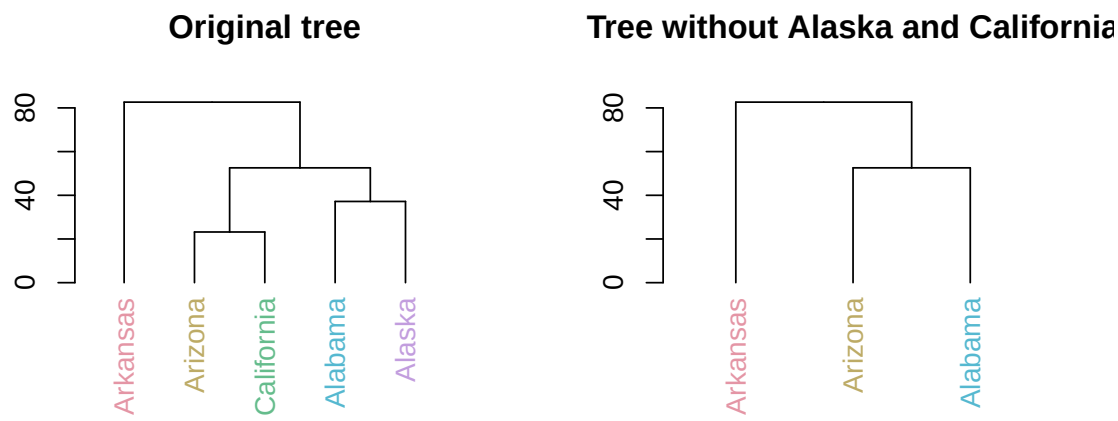


3.4. Trimming leaves

Trimming a tree from some leaves can be done using the `trim` (S3 method) function (notice that the attributes of the trimmed tree are updated):

```
hc <- hclust(dist(USArrests[1:5, ]), "ave")
dend <- as.dendrogram(hc)
library(colorspace)
labels_colors(dend) <- rainbow_hcl(5)

trimmed_dend <- trim(dend, c("Alaska", "California"))

str(unclass(trimmed_dend))

## List of 2
##  $ : atomic [1:1] 4
##   ..- attr(*, "members")= int 1
##   ..- attr(*, "height")= num 0
##   ..- attr(*, "label")= chr "Arkansas"
##   ..- attr(*, "leaf")= logi TRUE
##   ..- attr(*, "nodePar")=List of 2
##   .. ..$ lab.col: chr "#E495A5"
##   .. ..$ pch    : logi NA
##  $ :List of 2
##   ..$ : atomic [1:1] 3
##   .. ..- attr(*, "label")= chr "Arizona"
##   .. ..- attr(*, "members")= int 1
##   .. ..- attr(*, "height")= num 0
##   .. ..- attr(*, "leaf")= logi TRUE
##   .. ..- attr(*, "nodePar")=List of 2
##   .. .. ..$ lab.col: chr "#BDAB66"
##   .. .. ..$ pch    : logi NA
```

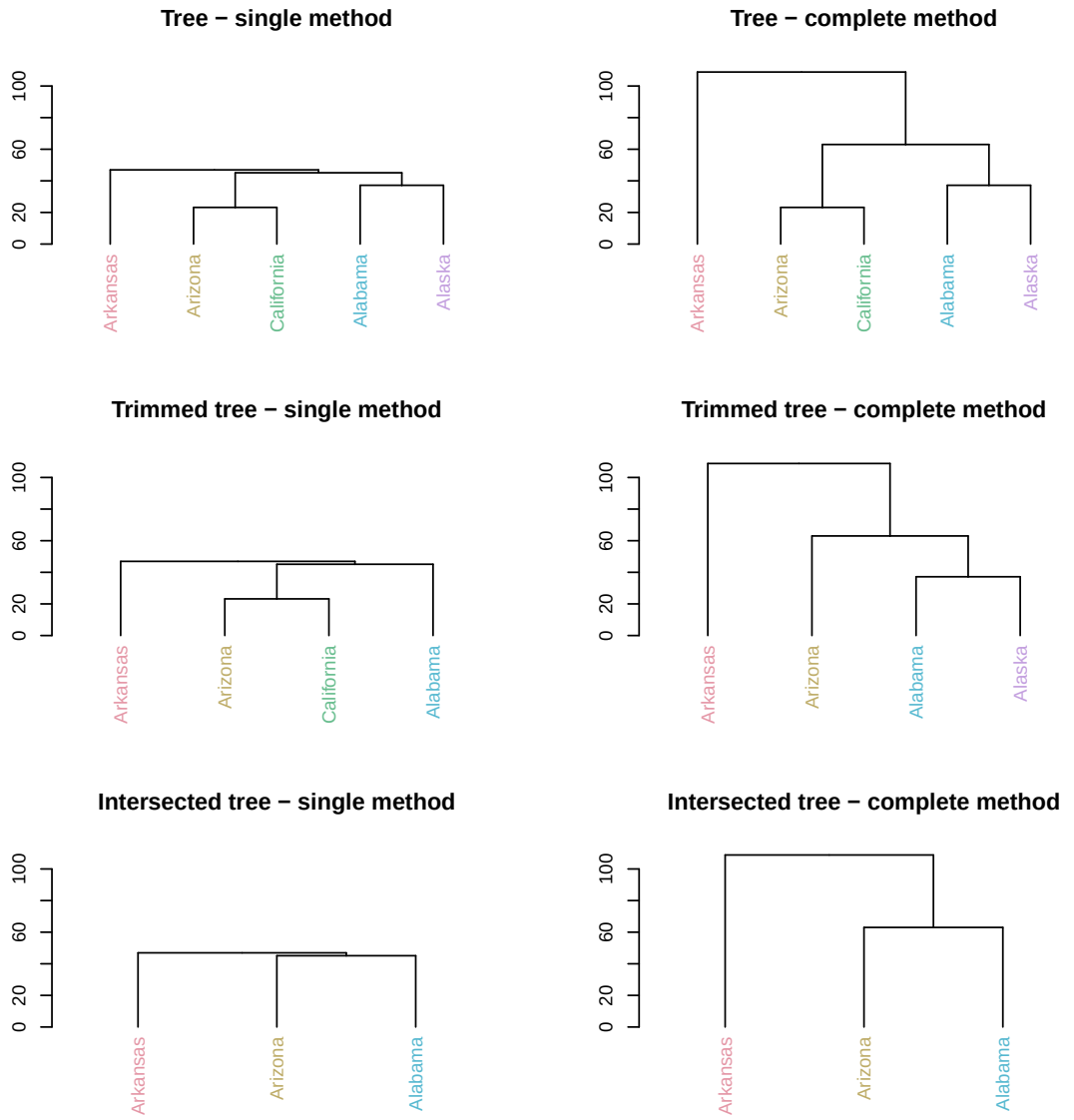
```
##   ..$ : atomic [1:1] 1
##   .. ..- attr(*, "label")= chr "Alabama"
##   .. ..- attr(*, "members")= int 1
##   .. ..- attr(*, "height")= num 0
##   .. ..- attr(*, "leaf")= logi TRUE
##   .. ..- attr(*, "nodePar")=List of 2
##   .. .. ..$ lab.col: chr "#55B8D0"
##   .. .. ..$ pch    : logi NA
##   ..- attr(*, "members")= num 2
##   ..- attr(*, "midpoint")= num 0.5
##   ..- attr(*, "height")= num 52.6
##  - attr(*, "members")= num 3
##  - attr(*, "midpoint")= num 0.75
##  - attr(*, "height")= num 82.6

par(mfrow = c(1, 2))
plot(dend, main = "Original tree")
plot(trimmed_dend, main = "Tree without Alaska and California")
```



If we have two trees, we can use the `intersect_trees` function to reduce both trees to have the same labels (this will be useful later when we'd like to compare the two trees):

```
hc_1 <- hclust(dist(USArrests[1:5, ]), "single")
hc_2 <- hclust(dist(USArrests[1:5, ]), "complete")
dend_1 <- as.dendrogram(hc_1)
dend_2 <- as.dendrogram(hc_2)

library(colorspace)
labels_colors(dend_1) <- rainbow_hcl(5)
labels_colors(dend_2) <- rainbow_hcl(5)
```

```
trimmed_dend_1 <- trim(dend_1, c("Alaska"))
trimmed_dend_2 <- trim(dend_2, c("California"))

dends_12 <- intersect_trees(trimmed_dend_1, trimmed_dend_2)

par(mfrow = c(3, 2))
plot(dend_1, main = "Tree - single method", ylim = c(0, 110))
plot(dend_2, main = "Tree - complete method", ylim = c(0, 110))
plot(trimmed_dend_1, main = "Trimmed tree - single method", ylim = c(0,
    110))
plot(trimmed_dend_2, main = "Trimmed tree - complete method",
    ylim = c(0, 110))
plot(dends_12[[1]], main = "Intersected tree - single method",
    ylim = c(0, 110))
plot(dends_12[[2]], main = "Intersected tree - complete method",
    ylim = c(0, 110))
```



Sidenote: a similar function, called `plotColoredClusters`, is available in the **ClassDiscovery** package for `hclust` objects.

3.5. Rotating branches

A dendrogram is an object which can be rotated on its hinges without changing its topological.

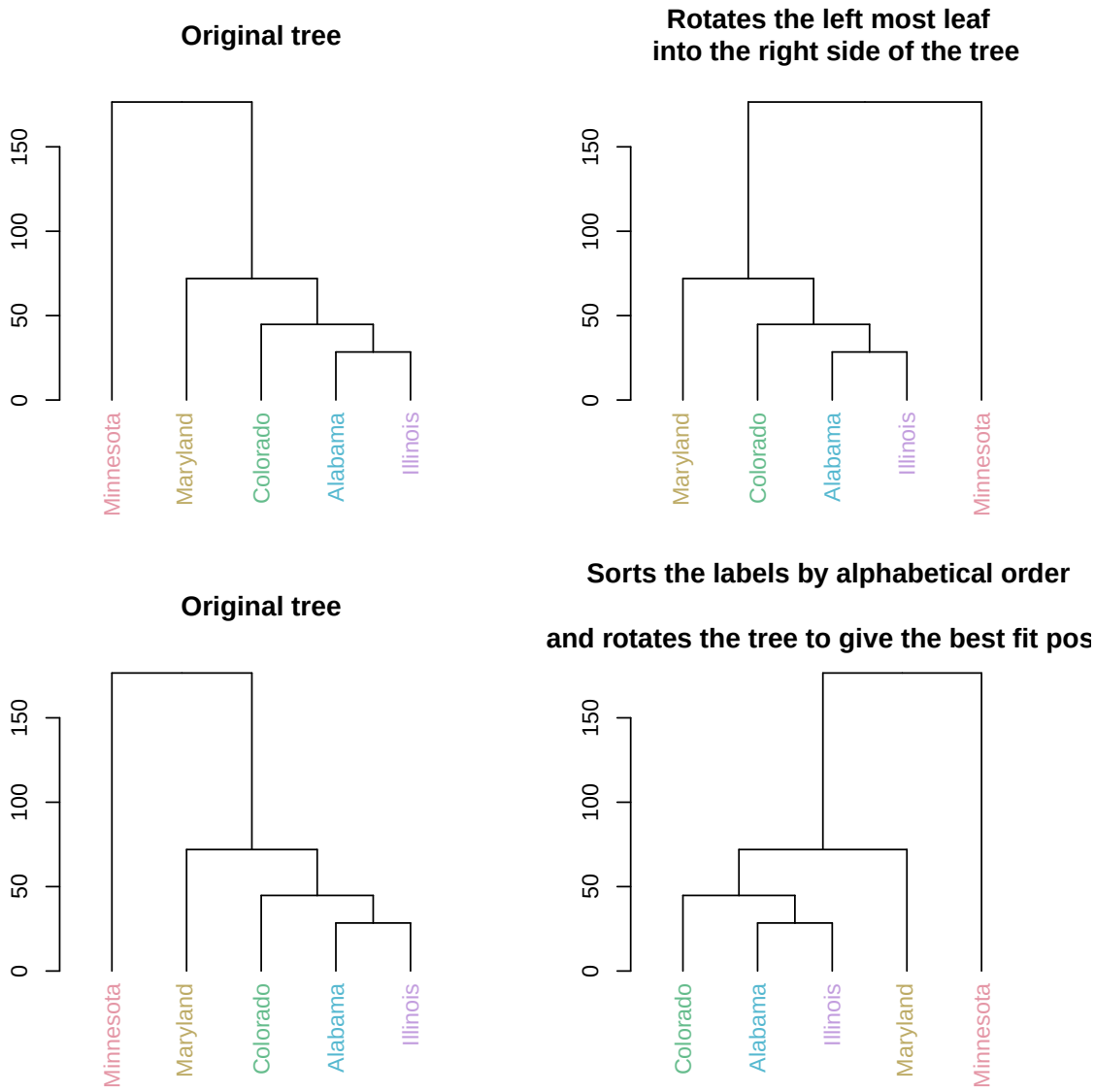
Rotating a dendrogram in base R can be done using the `reorder` function. The problem with this function is that it is not very intuitive. For this reason we wrote the `rotate` function. It has two main arguments: the object, and the order we wish to rotate it by. The order parameter can be either a numeric vector, used in a similar way we would order a simple

character vector. Or, the order parameter can also be a character vector of the labels of the tree, given in the new desired order of the tree.

It is also worth noting that some order are impossible to achieve for a given tree's topology. In such a case, the function will do its "best" to get as close as possible.

Here are a few examples:

```
hc <- hclust(dist(USArrests[c(1, 6, 13, 20, 23), ]), "ave")
dend <- as.dendrogram(hc)

# For dendrogram objects:
require(colorspace)
labels_colors(dend) <- rainbow_hcl(nleaves(dend))
# let's color the labels to make the followup of the rotation
# easier
par(mfrow = c(2, 2))
plot(dend, main = "Original tree")
plot(rotate(dend, c(2:5, 1)), main = "Rotates the left most leaf \n into the right side of
plot(dend, main = "Original tree")
plot(sort(dend), main = "Sorts the labels by alphabetical order \n \n     and rotates the
```



3.6. Cutting trees

The `hclust` function comes with a very powerfull `cutree` function, for extracting cluster grouping of the original data based on cutting the heirarchical tree at some height (or setting a predefined k - number of clusters). The limitation of this function is that it is only available for `hclust` object. Hence, if we are dealing with a tree which is NOT an ultrametric tree (e.g: ultrametric tree = a tree with monotone clustering heights), `cutree` would not be available for us (since `as.hclust` would not work on our dendrogram).

In **dendextend**, we extend `cutree` by turning it into an S3 method, with methods for dendrogram and phylo objects. The phylo method is only turning the phylo object to an hclust, and tries to work on it there. However, the dendrogram method (`cutree.dendrogram`) is a complete re-writing of `cutree` based on the `cut.dendrogram` function. `cutree.dendrogram` fully emulates `cutree` by default, but at the same time extends it with the type of trees it

can handle, and with some other options.

Since `cutree.dendrogram` is written in R, it is slower than `cutree` which is implemented in C. If we can turn the dendrogram into hclust, we will use `cutree.hclust`, otherwise - `cutree.dendrogram` will be used.

Here are several examples of how these functions are used:

```
hc <- hclust(dist(USArrests[c(1, 6, 13, 20, 23), ]), "ave")
dend <- as.dendrogram(hc)
unbranch_dend <- unbranch(dend, 2)

cutree(hc, k = 2:4)  # on hclust

##           2 3 4
## Alabama   1 1 1
## Colorado  1 1 2
## Illinois  1 1 1
## Maryland  1 2 3
## Minnesota 2 3 4

cutree(dend, k = 2:4)  # on dendrogram

##           2 3 4
## Alabama   1 1 1
## Colorado  1 1 2
## Illinois  1 1 1
## Maryland  1 2 3
## Minnesota 2 3 4

cutree(hc, k = 2)  # on hclust

##   Alabama  Colorado  Illinois  Maryland Minnesota
##         1         1         1         1         2

cutree(dend, k = 2)  # on dendrogram

##   Alabama  Colorado  Illinois  Maryland Minnesota
##         1         1         1         1         2

cutree(dend, h = c(20, 25.5, 50, 170))

##           20 25.5 50 170
## Alabama    1    1  1   1
## Colorado   2    2  1   1
## Illinois   3    3  1   1
## Maryland   4    4  2   1
## Minnesota  5    5  3   2
```

```
cutree(hc, h = c(20, 25.5, 50, 170))

##           20 25.5 50 170
## Alabama    1    1  1   1
## Colorado   2    2  1   1
## Illinois   3    3  1   1
## Maryland   4    4  2   1
## Minnesota  5    5  3   2

# the default (ordered by original data's order)
cutree(dend, k = 2:3, order_clusters_as_data = FALSE)

## [1] 2 1 1 1 1

labels(dend)

## [1] "Minnesota" "Maryland"  "Colorado"  "Alabama"
## [5] "Illinois"

# cutree now works for unbranched trees!
# as.hclust(unbranch_dend) # ERROR - can not do this...
cutree(unbranch_dend, k = 2)  # all NA's

## Warning:  Couldn't cut the tree - returning NA.
## Warning:  You (probably) have some branches with equal heights so that there
exist no height(h) that can create 2  clusters

## [1] NA NA NA NA NA

cutree(unbranch_dend, k = 1:4)

## Warning:  Couldn't cut the tree - returning NA.
## Warning:  You (probably) have some branches with equal heights so that there
exist no height(h) that can create 2  clusters

##           1  2 3 4
## Alabama   1 NA 2 2
## Colorado  1 NA 2 3
## Illinois  1 NA 2 2
## Maryland  1 NA 3 4
## Minnesota 1 NA 1 1

cutree(unbranch_dend, h = c(20, 25.5, 50, 170))
```

```
##           20 25.5 50 170
## Alabama    1    1  2   2
## Colorado   2    2  2   2
## Illinois   3    3  2   2
## Maryland   4    4  3   3
## Minnesota  5    5  1   1

require(microbenchmark)
## this shows how as.hclust is expensive - but still worth it
## if possible
microbenchmark(cutree(hc, k = 2:4), cutree(as.hclust(dend), k = 2:4),
    cutree(dend, k = 2:4), cutree(dend, k = 2:4, try_cutree_hclust = FALSE))

## Unit: microseconds
##                                                 expr    min
##                                  cutree(hc, k = 2:4)  136.6
##                       cutree(as.hclust(dend), k = 2:4) 1727.9
##                                 cutree(dend, k = 2:4) 1825.9
##  cutree(dend, k = 2:4, try_cutree_hclust = FALSE) 8218.0
##      lq median     uq     max neval
##   142.2  145.6  194.6   324.8   100
##  1755.9 1834.9 1902.6  4882.0   100
##  1889.8 1945.5 1994.2  4950.9   100
##  8372.6 8512.6 8707.1 11539.5   100

# the dendrogram is MUCH slower...


# and trying to 'hclust' is not expensive (which is nice...)
microbenchmark(cutree_unbranch_dend = cutree(unbranch_dend, k = 2:4,
    warn = FALSE), cutree_unbranch_dend_not_trying_to_hclust = cutree(unbranch_dend,
    k = 2:4, try_cutree_hclust = FALSE, warn = FALSE))

## Unit: milliseconds
##                                         expr   min    lq
##                         cutree_unbranch_dend 6.691 6.784
##  cutree_unbranch_dend_not_trying_to_hclust 6.283 6.375
##  median    uq    max neval
##   6.894 7.052  9.978   100
##   6.488 6.679 13.087   100
```

Having `cutree.dendrogram` available to us, we can now gain the ability to color branches for trees which can not be represented as an `hclust` object.

3.7. Coloring branches

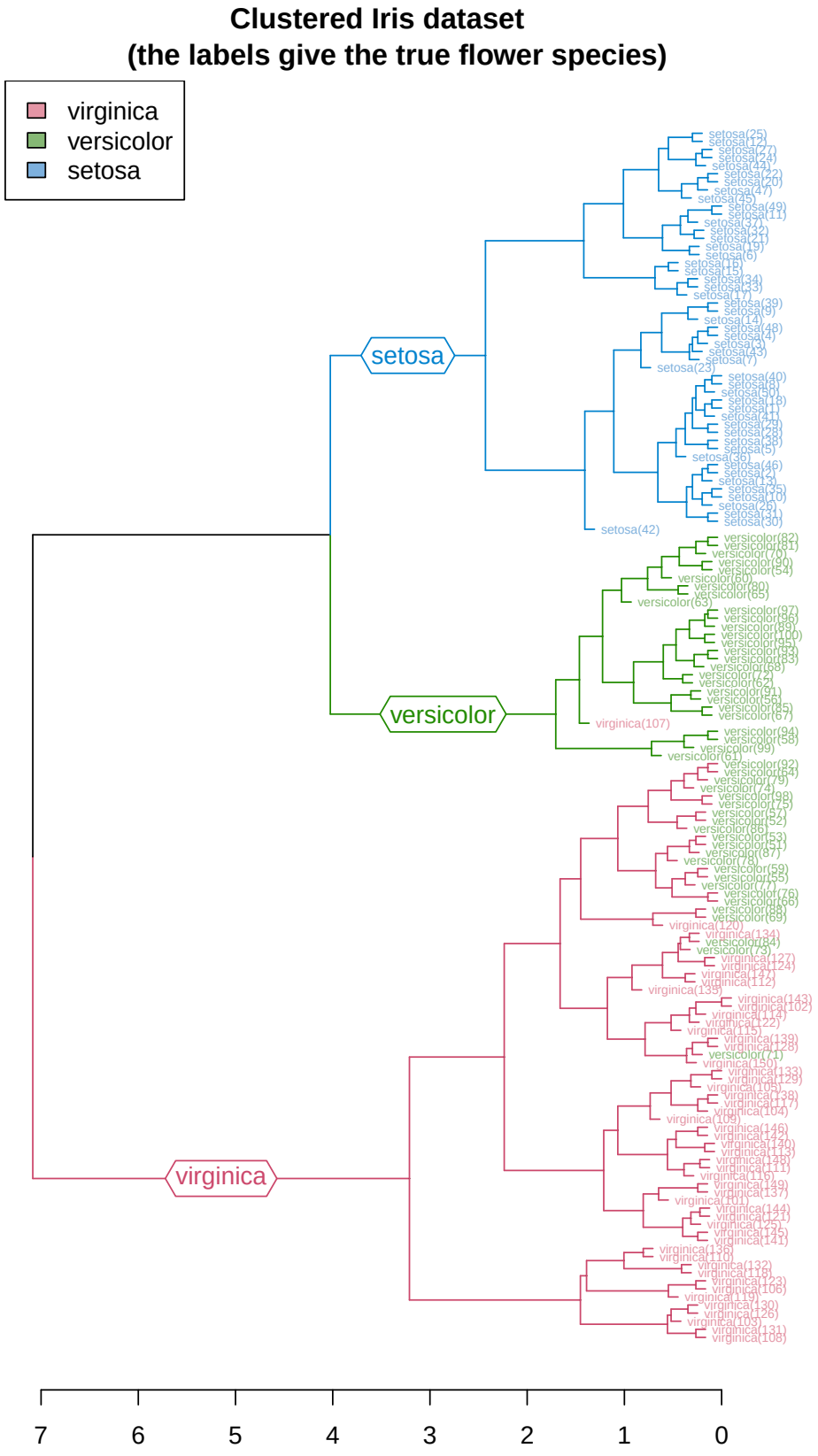
Dendrogram plots with colored branches are used to easly distinguishing between different clusters on a tree. They have been available in R for many years in threads on the mailing lists and through various package. However, until recently, all of the functions in packages have always given the user a new `plot` function, without seperating the coloring of branches of a dendrogram from its plotting. Often the function for actually plotting the colored branched dendrogram would be hidden from the user. For example, the **labeltodendro** package (Nia and Stephens 2011) gives a colored branch plot through the `colorplot` function, but the work horse for this is available in a hidden function called `dendroploth` or `dendroplotv`, both take care of the plotting by themselves (instead of modifying the dendrogram object, and then letting the base R function do the work). The same story happens in the **Heatplus** (Ploner 2012), where the `plot.annHeatmap2` function actually uses the hidden function `cutplot.dendrogram` for doing the plotting.

This was changed in the beginning of 2013 thanks to Gregory Jefferis's **dendroextras** package (Jefferis 2013), which orgranized this through the `colour_clusters` function. In the **dendextend** package we imported his code into a new function called `color_branches`, with several useful modifications on the original code. The biggest limitation in the `colour_clusters` function is that it relies on changing the `dendrogram` into `hclust` in order to use `cutree` on it and get the clusters. This has the advantage of being fast, but the **disadvantage** of restricting the function to ultrametric and binary trees (the type of tree `hclust` objects can handle). In our implementation we implemented a new function called `cutree.dendrogram` (available through an S3 method), which can either relly on `hclust` for speed, but in case it is not possible will revert to find the clustering on the dendrogram object itself (the function is completely compatible with the base R `cutree` function). Also, our implementation of the `color_branches` function allows for repeated-multilayered coloring the branches. Another potential bug which is avoided in our implementation is to handle cases where the labels of the tree are of type Integer instead of Character (this happens if the `dendrogram` came from an `hclust`, used on the `dist` of data that has no `rownames`).

Here is an example of using `color_branches` combined with `labels_colors`:

```
require(colorspace)

data(iris)
d_iris <- dist(iris[, -5])  # method='man' # is a bit better
hc_iris <- hclust(d_iris)
labels(hc_iris)  # no labels, because 'iris' has no row names

## character(0)

dend_iris <- as.dendrogram(hc_iris)
is.integer(labels(dend_iris))  # this could cause problems...

## [1] TRUE

iris_species <- rev(levels(iris[, 5]))
dend_iris <- color_branches(dend_iris, k = 3, groupLabels = iris_species)
is.character(labels(dend_iris))  # labels are no longer 'integer'
```

```
## [1] TRUE

# have the labels match the real classification of the
# flowers:
labels_colors(dend_iris) <- rainbow_hcl(3)[sort_levels_values(as.numeric(iris[,
    5])[order.dendrogram(dend_iris)])]

# We'll add the flower type
labels(dend_iris) <- paste(as.character(iris[, 5])[order.dendrogram(dend_iris)],
    "(", labels(dend_iris), ")", sep = "")

dend_iris <- hang.dendrogram(dend_iris, hang_height = 0.1)

# reduce the size of the labels:
dend_iris <- assign_values_to_leaves_nodePar(dend_iris, 0.5,
    "lab.cex")

## Warning:  Length of value vector was shorter than the number of leaves - vector
value recycled

par(mar = c(3, 3, 3, 7))
plot(dend_iris, main = "Clustered Iris dataset\n     (the labels give the true flower spec
    horiz = TRUE, nodePar = list(cex = 0.007))
legend("topleft", legend = iris_species, fill = rainbow_hcl(3))
```

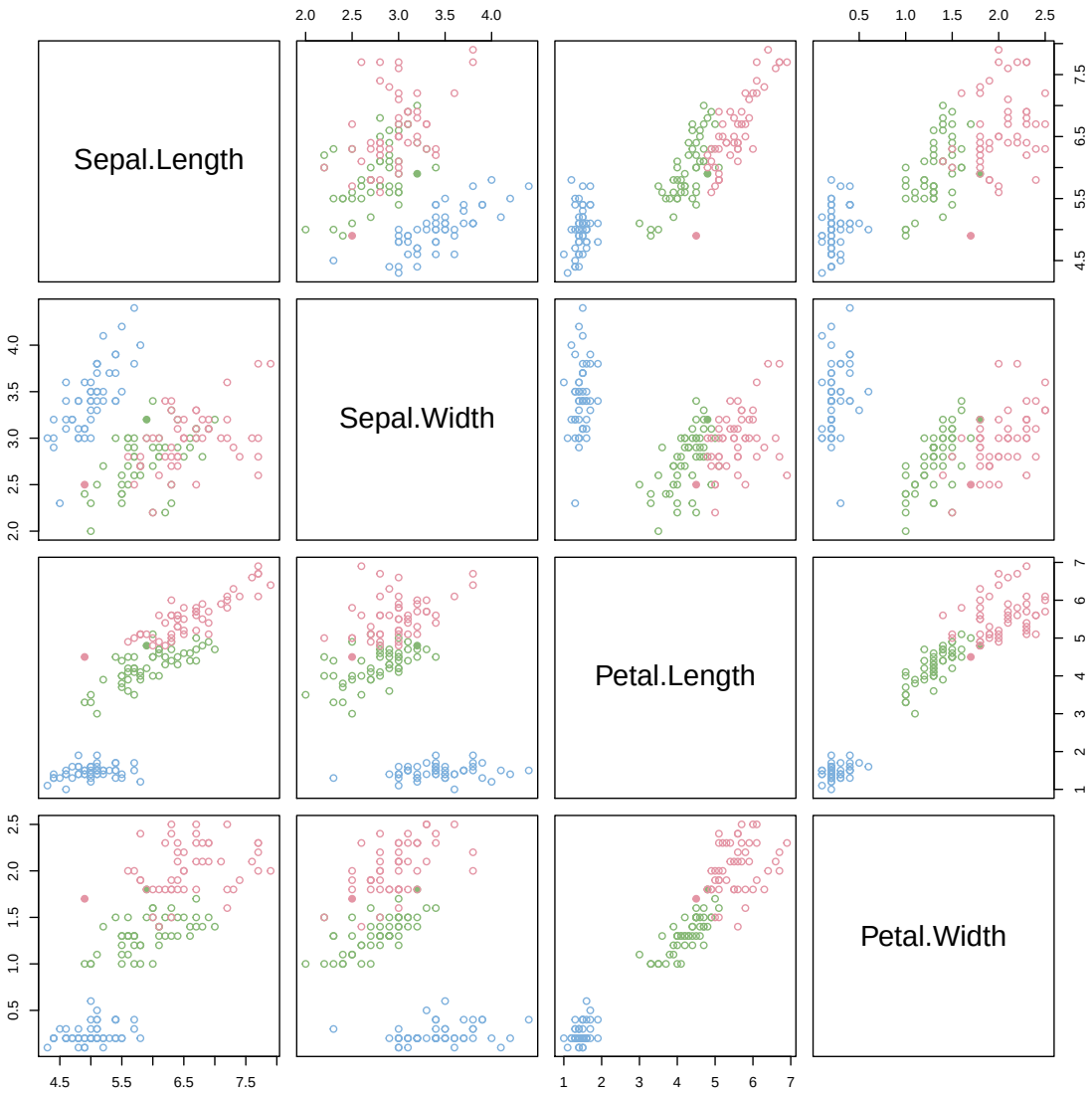

Clustered Iris dataset
(the labels give the true flower species)
virginica
versicolor
setosa
setosa
versicolor
virginica
7
6
5
4
3
2
1
0

This simple visualization easily demonstrates how the seperation of the heirarchical clustering is very good with the "setosa" sepcies, but misses in labeling many "versicolor" species as "virginica".

The hanging of the tree also helps to locate extreme observations. For example, we can see that observation "virginica (107)" is not very similar to the Versicolor species, but still, it is amoung them. Also, "Versicolor (71)" is too much "within" the Virginica bush, and it is a wonder why that is. Of course, the Iris dataset is very well known, and simpler pairs plot often help to locate such problems, yet - dendrogram trees (with all of their limitations) can help gain insights for very high-dimensional data where a simple pairs plot is not possible.

Here is the similar pairs plot. Notice the filled red and green dots, corrosponding with the extreme observations we have noticed in the dendrogram. Petal.Length vs Petal.Width gives the best example of why the green dot is an outlier. Whereas most plots show why the red dot is an outlier. There are of course other outliers, and a problem with classification, which we will not go futher into.

```
require(colorspace)

species_col <- rev(rainbow_hcl(3))[as.numeric(iris[, 5])]
# species_col[c(107,71)] <- 'red' # highlight the two extreme
# cases we mentioned
species_pch <- rep(1, length(species_col))
species_pch[c(107, 71)] <- 19  # highlight the two extreme cases we mentioned

pairs(iris[, -5], col = species_col, pch = species_pch)
```



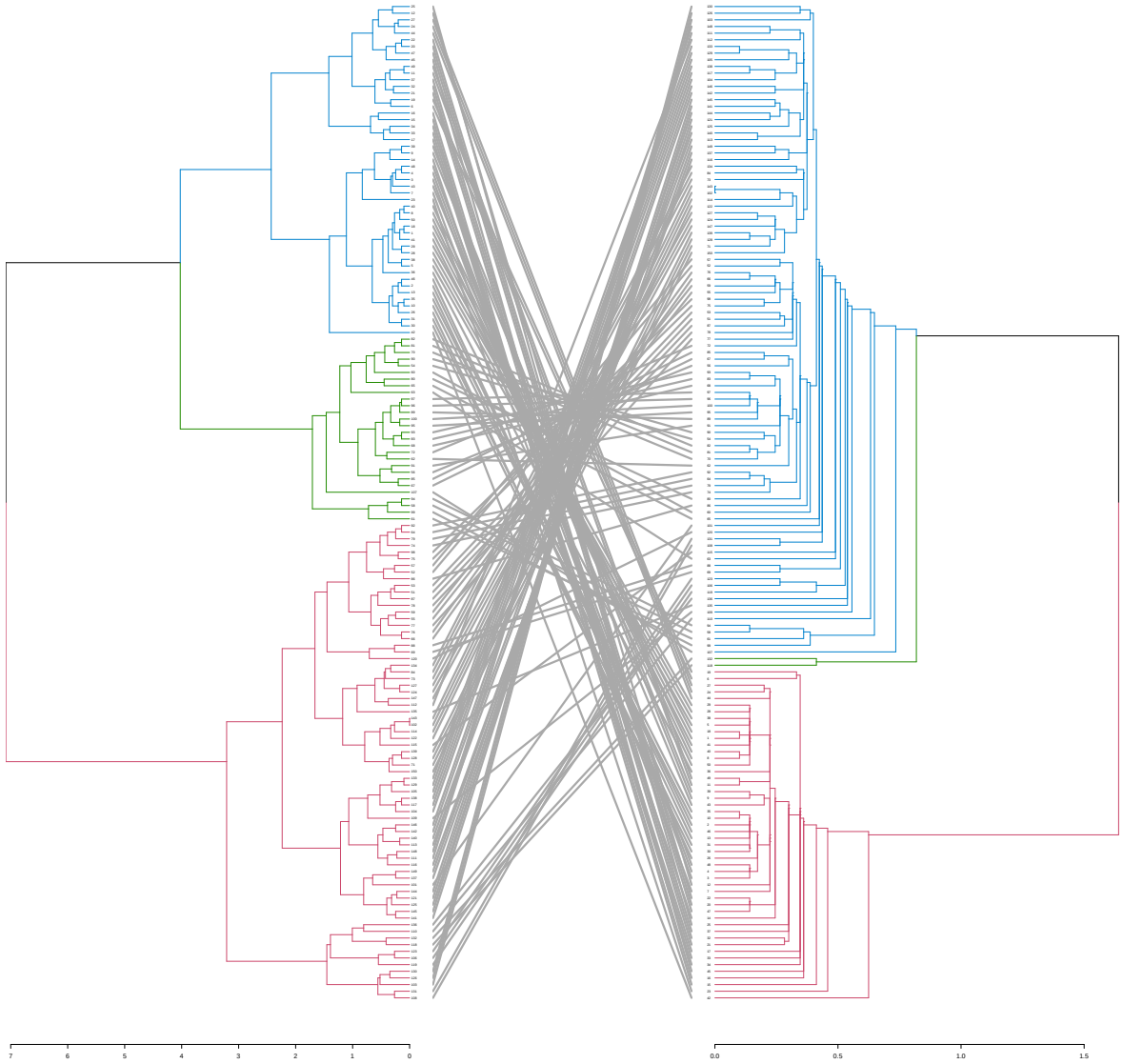
4. Tanglegrams - visually comparing two trees side-by-side

4.1. Tanglegram visualization

A tanglegram plot gives two dendrogram (with the same set of labels), one facing the other, and having their labels connected by lines. Tanglegram can be used for visually comparing two methods of heirarchical clustering, and are sometimes used in biology when comparing two phylogenetic trees.

Here is an example of creating a tanglegram using **dendextend**:

```
hc1 <- hclust(dist(iris[, -5]), "com")
hc2 <- hclust(dist(iris[, -5]), "single")
dend1 <- as.dendrogram(hc1)
dend2 <- as.dendrogram(hc2)
tanglegram(dend1, dend2, lab.cex = 0.5, edge.lwd = 1, margin_inner = 3,
    type = "r", center = TRUE, k_branches = 3)

## Warning:  Length of value vector was shorter than the number of leaves - vector
value recycled
## Warning:  Length of value vector was shorter than the number of leaves - vector
value recycled
```



Notice that since every manipulation of the trees require going through all of their nodes, it

might be better (when comparing large trees), to do all of the manipulations upfront, and only then plot the tanglegram.

We can also notice the mess of tangled lines of the two trees, a mess we wish to untangle.

4.2. Measuring Entanglement

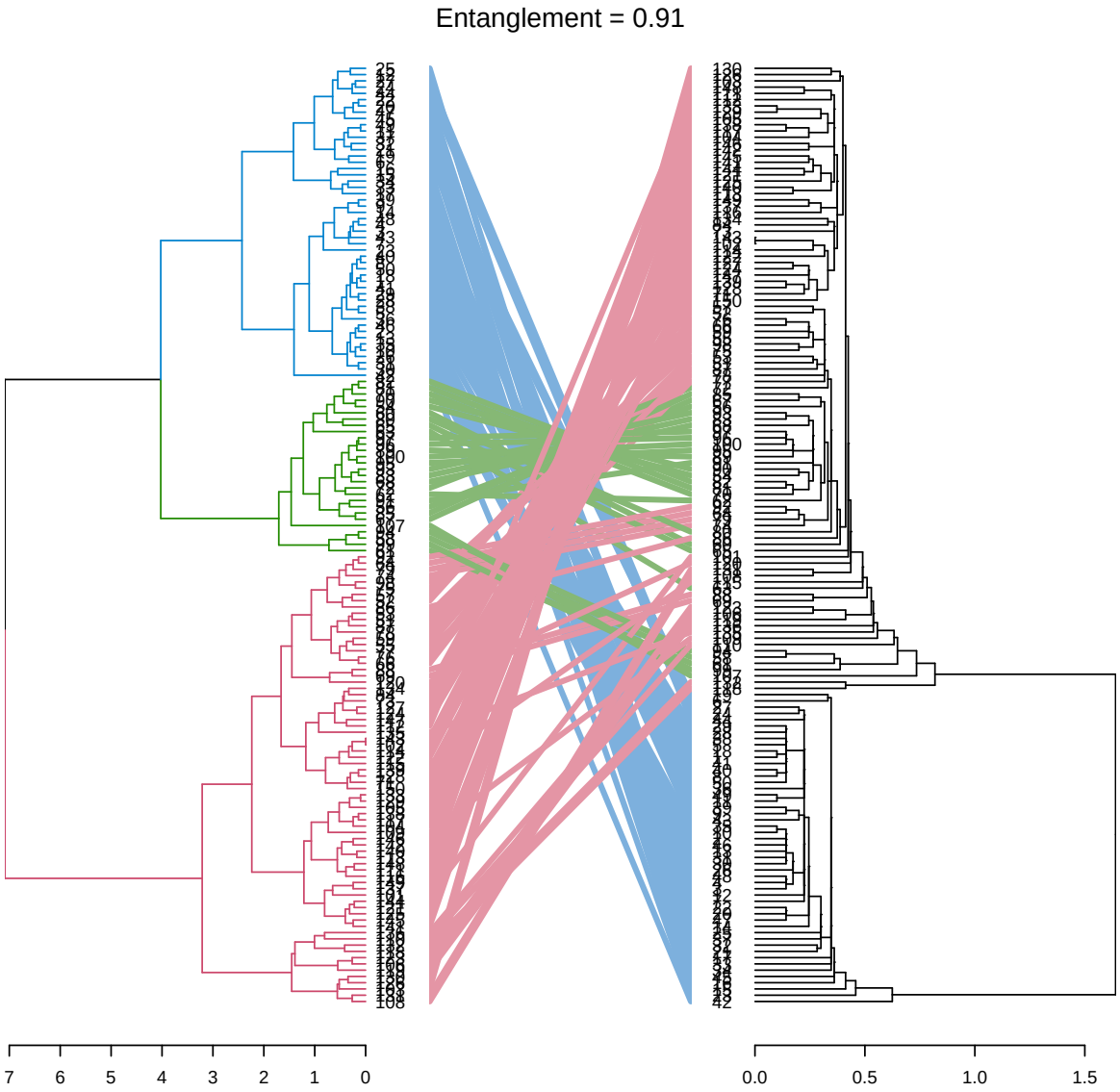
When comparing two trees via a tanglegram, we can use the `entanglement` function in order to measure how much the two trees are "aligned".

Entanglement is measured by giving the left tree's labels the values of 1 till tree size, and than match these numbers with the right tree. Now, entanglement is the L norm distance between these two vectors. That is, we take the sum of the absolute difference (each one in the power of L). e.g: `sum(abs(x-y)**L)`. And this is devided by the "worst case" entanglement level (e.g: when the right tree is the complete reverse of the left tree).

L tells us which panelty level we are at (L0, L1, L2, partial L's etc). L>1 means that we give a big panelty for sharp angles. While L->0 means that any time something is not a streight horizontal line, it gets a large penalty If L=0.1 it means that we much prefer streight lines over non streight lines

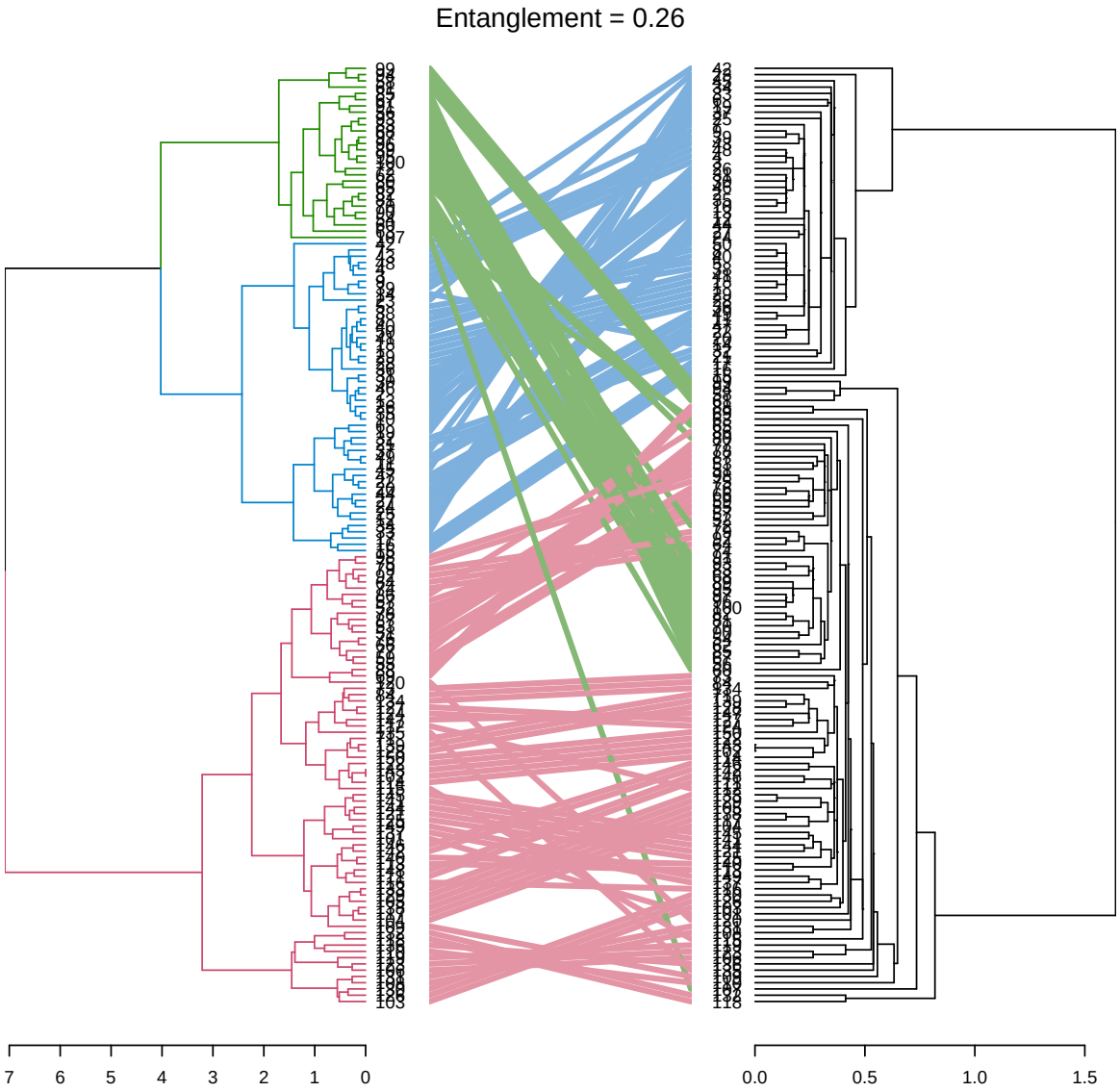
For example:

```
hc1 <- hclust(dist(iris[, -5]), "com")
hc2 <- hclust(dist(iris[, -5]), "single")
dend1 <- as.dendrogram(hc1)
dend2 <- as.dendrogram(hc2)
dend1 <- color_branches(dend1, 3)
# dend2 <- color_branches(dend2, 3) # colors will not match
# nicely
labels(dend2) <- as.character(labels(dend2))

require(colorspace)

col_lines_by_left_groups <- rainbow_hcl(3)[cutree(dend1, 3, order_clusters_as_data = FALSE
    sort_cluster_numbers = TRUE)]
tanglegram(dend1, dend2, color_lines = col_lines_by_left_groups,
    main = paste("Entanglement =", round(entanglement(dend1,
        dend2), 2)))
```



And here is the entanglement of a tiny bit prettier tanglegram:

```
s_dend1 <- sort(dend1)
s_dend2 <- sort(dend2)

# s_dend1 <- color_branches(s_dend1, 3) # If I don't do this
# again I will get different colors
col_lines_by_left_groups_sorted <- col_lines_by_left_groups[match(labels(s_dend1),
    labels(dend1))]
tanglegram(s_dend1, s_dend2, color_lines = col_lines_by_left_groups_sorted,
    main = paste("Entanglement =", round(entanglement(s_dend1,
        s_dend2), 2)))
```



4.3. Finding an optimal rotation

Random search

Finding an optimal rotation for the tanglegram of two dendrogram is a hard problem.

This problem is also harder for larger trees, so let us pick a tree from only 30 flowers, and look at comparing two clustering methods, complete and single:

```
set.seed(51324626)
ss <- sample(1:150, size = 30)
hc1 <- hclust(dist(iris[ss, -5]), "com")
hc2 <- hclust(dist(iris[ss, -5]), "single")
dend1 <- as.dendrogram(hc1)
```

```
dend2 <- as.dendrogram(hc2)
dend1 <- color_branches(dend1, 3)
# dend2 <- color_branches(dend2, 3) # colors will not match
# nicely
labels(dend2) <- as.character(labels(dend2))

require(colorspace)

col_lines_by_left_groups <- rainbow_hcl(3)[cutree(dend1, 3, order_clusters_as_data = FALSE
    sort_cluster_numbers = TRUE)]
tanglegram(dend1, dend2, color_lines = col_lines_by_left_groups,
    main = paste("Entanglement =", round(entanglement(dend1,
        dend2), 2)))
```

One solution for improving the tanglegram would be to randomely search the rotated tree space for a better solution.

For example:

```
set.seed(65168)
dend12 <- untangle_random_search(dend1, dend2, R = 100)
tanglegram(dend12[[1]], dend12[[2]], color_lines = col_lines_by_left_groups,
    main = paste("Entanglement =", round(entanglement(dend12[[1]],
        dend12[[2]]), 2)))
```

We can see we already got something better. An advantage of the random search is the ability to create many many trees and compare them to find the best pair.

A one-side greedy search

Let's use a greedy forward step wise rotation of the right tree, to see if we can find a better solution for comparing the two trees. Notice that this may take some time to run (the larger the tree, the longer it would take), but we can limit the search for smaller k's, and see what improvement that can bring us:

```
best_dend_heights_per_k <- heights_per_k.dendrogram(dend2)

dend2_2 <- untangle_step_rotate_1side(dend2, dend1, k_seq = 2:3,
    best_dend_heights_per_k = best_dend_heights_per_k)

## Loading required package:  plyr

# notice that here we are only using
tanglegram(dend1, dend2_2, color_lines = col_lines_by_left_groups,
    main = paste("Entanglement =", round(entanglement(dend1,
        dend2_2), 2)))
```

```
entanglement(dend1, dend2_2)

## [1] 0.06166
```

Let's try to go "all the way down" all possible k's:

```
dend2_3 <- untangle_step_rotate_1side(dend2, dend1, best_dend_heights_per_k = best_dend_he
tanglegram(dend1, dend2_3, color_lines = col_lines_by_left_groups,
    main = paste("Entanglement =", round(entanglement(dend1,
        dend2_3), 2)))
```

```
entanglement(dend1, dend2_3)

## [1] 0.04225
```

Combining a one-side greedy search with a random search

As we can see, the random search actually found something better than the forward selection search. But now we can use the forward search on the trees we found in the random search:

```
dend2_4 <- untangle_step_rotate_1side(dend12[[2]], dend12[[1]])
tanglegram(dend12[[1]], dend2_4, color_lines = col_lines_by_left_groups,
    main = paste("Entanglement =", round(entanglement(dend12[[1]],
        dend2_4), 2)))
```

```
entanglement(dend12[[1]], dend2_4)

## [1] 0
```

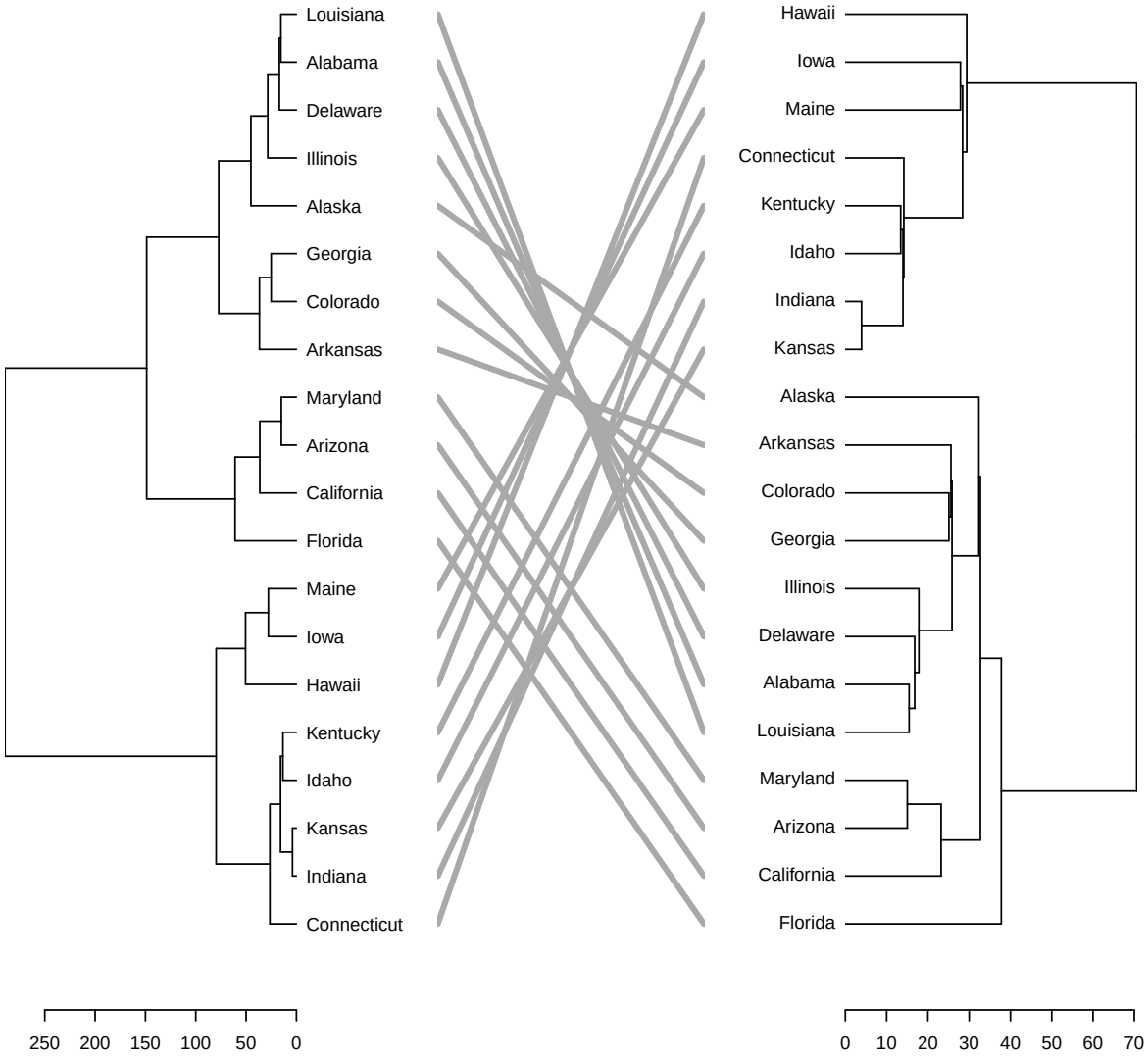
We see how this got us with a perfect solution. The trees look almost identical, but they are not (look at label 45 to see a leaf which is differently located between the two trees.). We can use the k_branches parameter for tanglegram, to help us see the difference between the two trees (notice labels 19 and 45

```
tanglegram(color_branches(dend12[[1]], col = 1, k = 1), dend2_4,
    color_lines = col_lines_by_left_groups, k_branches = 10,
    main = paste("Entanglement =", round(entanglement(dend12[[1]],
        dend2_4), 2)))
```

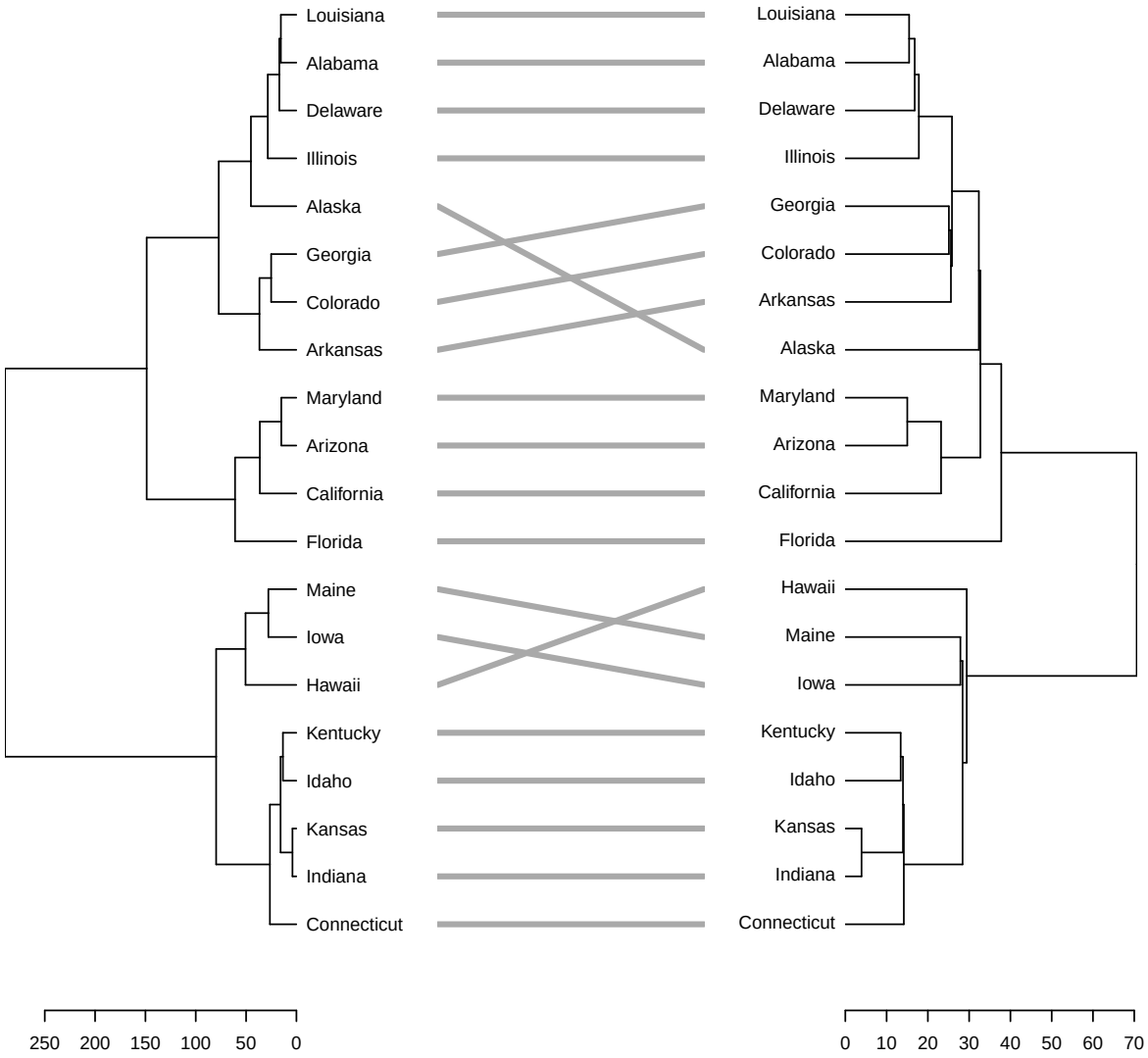
A two-side greedy search

Sometimes, using just one-sise greedy algorithm is not enough. Also, it is probably best to combine the random search with the two sided greedy stepwise search:

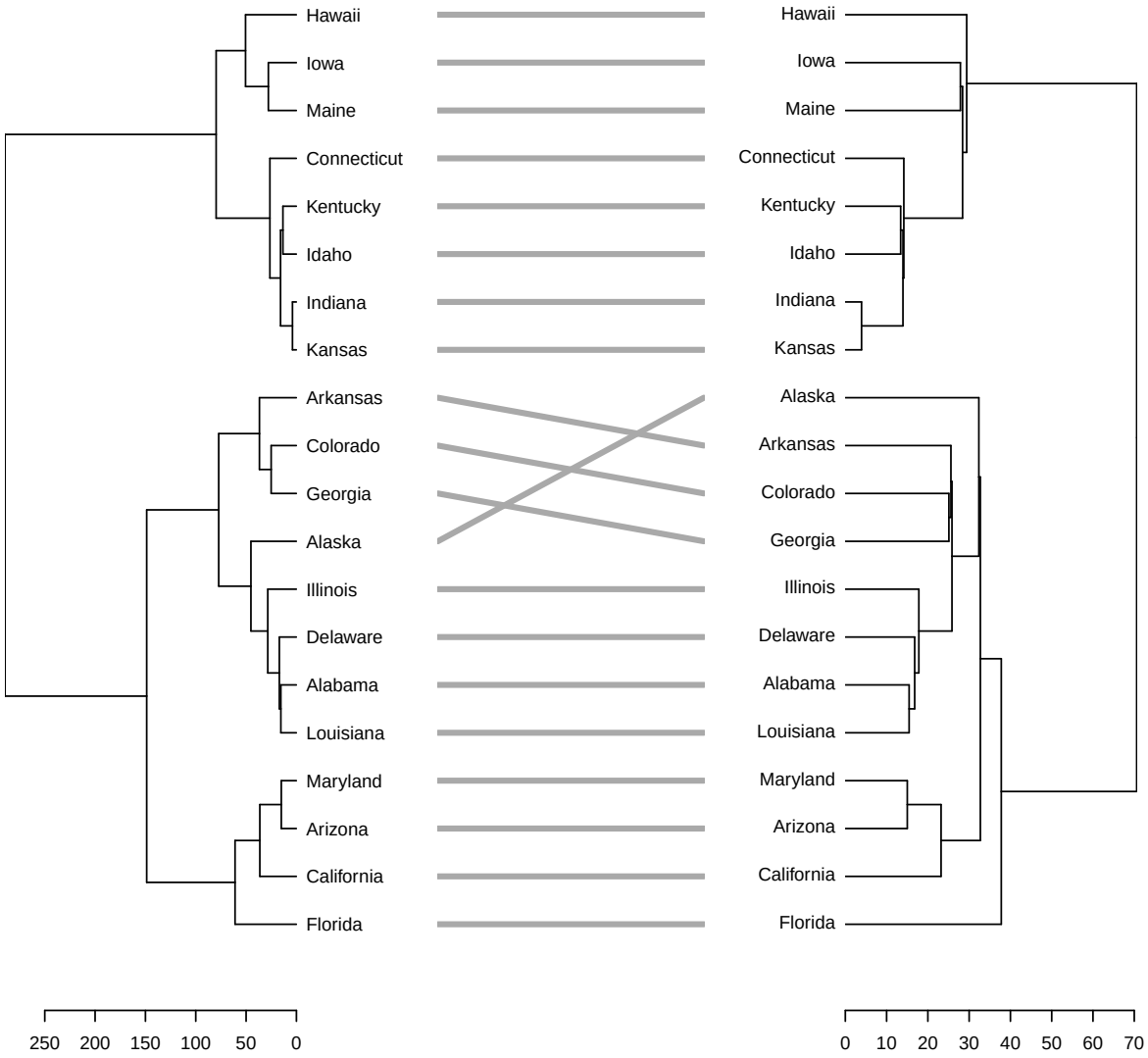
```
dend1 <- as.dendrogram(hclust(dist(USArrests[1:20, ])))
dend2 <- as.dendrogram(hclust(dist(USArrests[1:20, ]), method = "single"))
set.seed(3525)
dend2 <- shuffle(dend2)
tanglegram(dend1, dend2, margin_inner = 6.5)
```



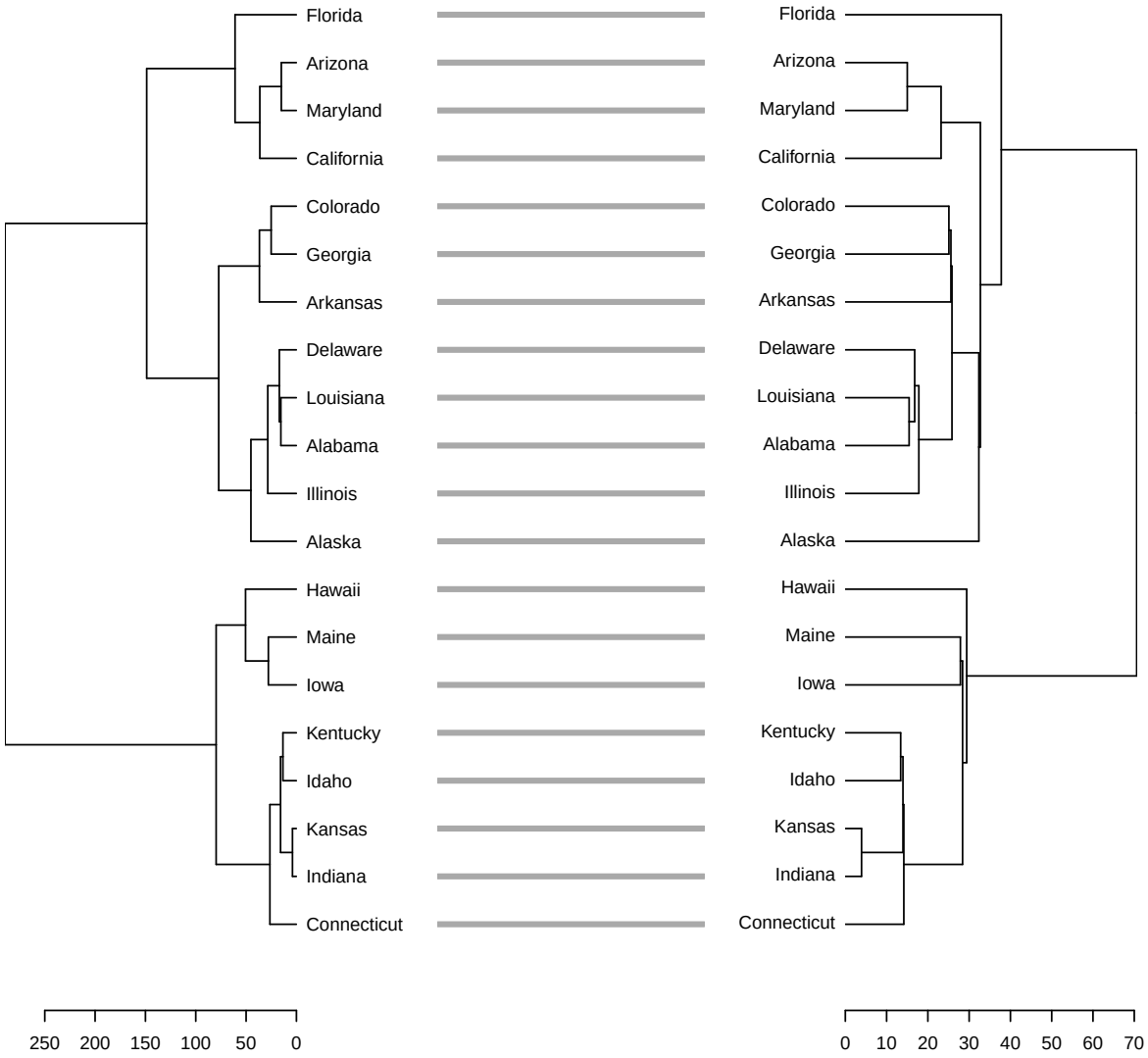

```
entanglement(dend1, dend2, L = 2)  # 0.79

## [1] 0.7917

dend2_corrected <- untangle_step_rotate_1side(dend2, dend1)
tanglegram(dend1, dend2_corrected, margin_inner = 6.5)  # Good.
```



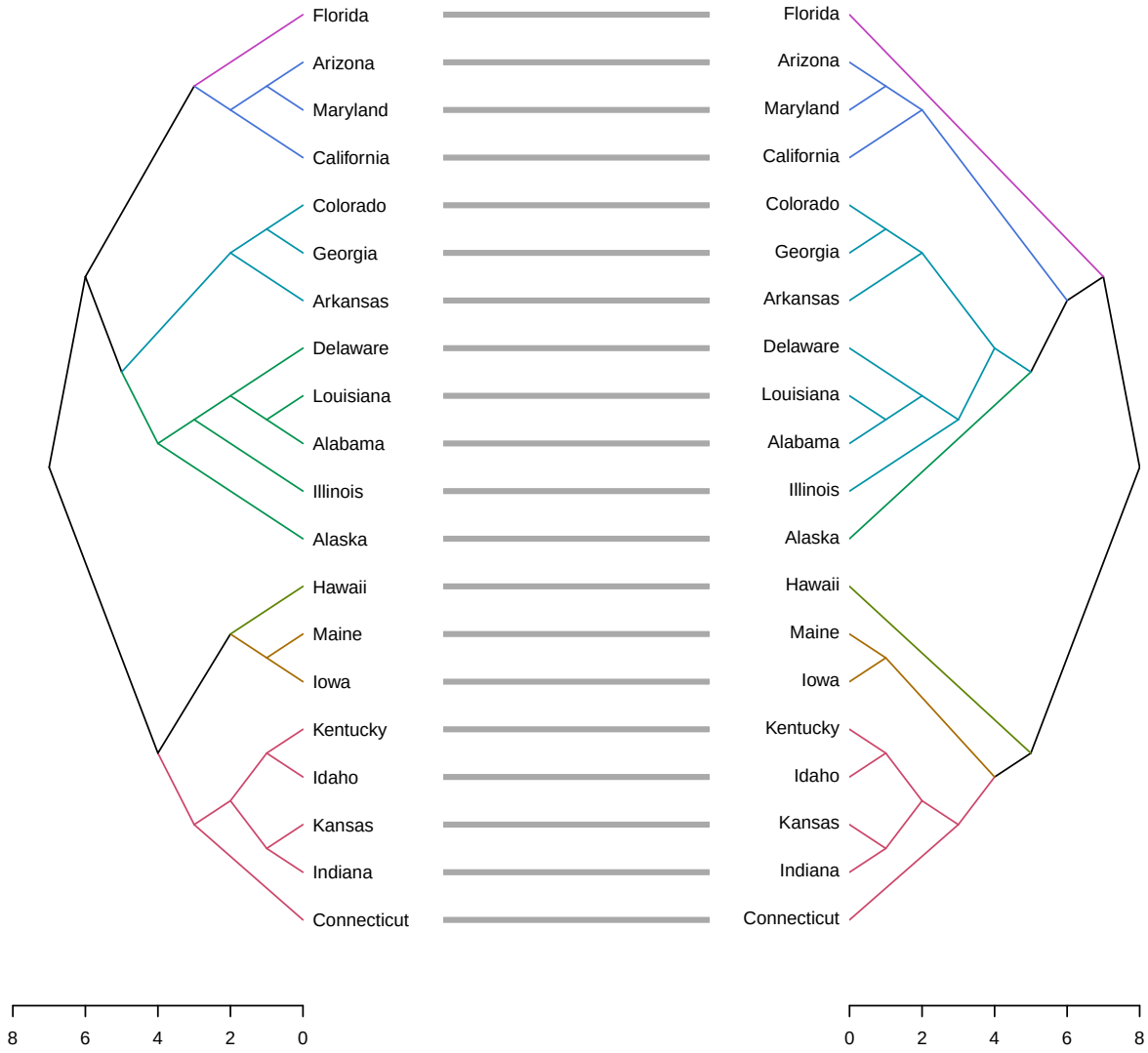

```
entanglement(dend1, dend2_corrected, L = 2)  # 0.0067

## [1] 0.006767

# it is better, but not perfect. Can we improve it?

dend12_corrected <- untangle_step_rotate_2side(dend1, dend2)

## We ran untangle  1  times

tanglegram(dend12_corrected[[1]], dend12_corrected[[2]], margin_inner = 6.5)  # Bet-
ter...
```



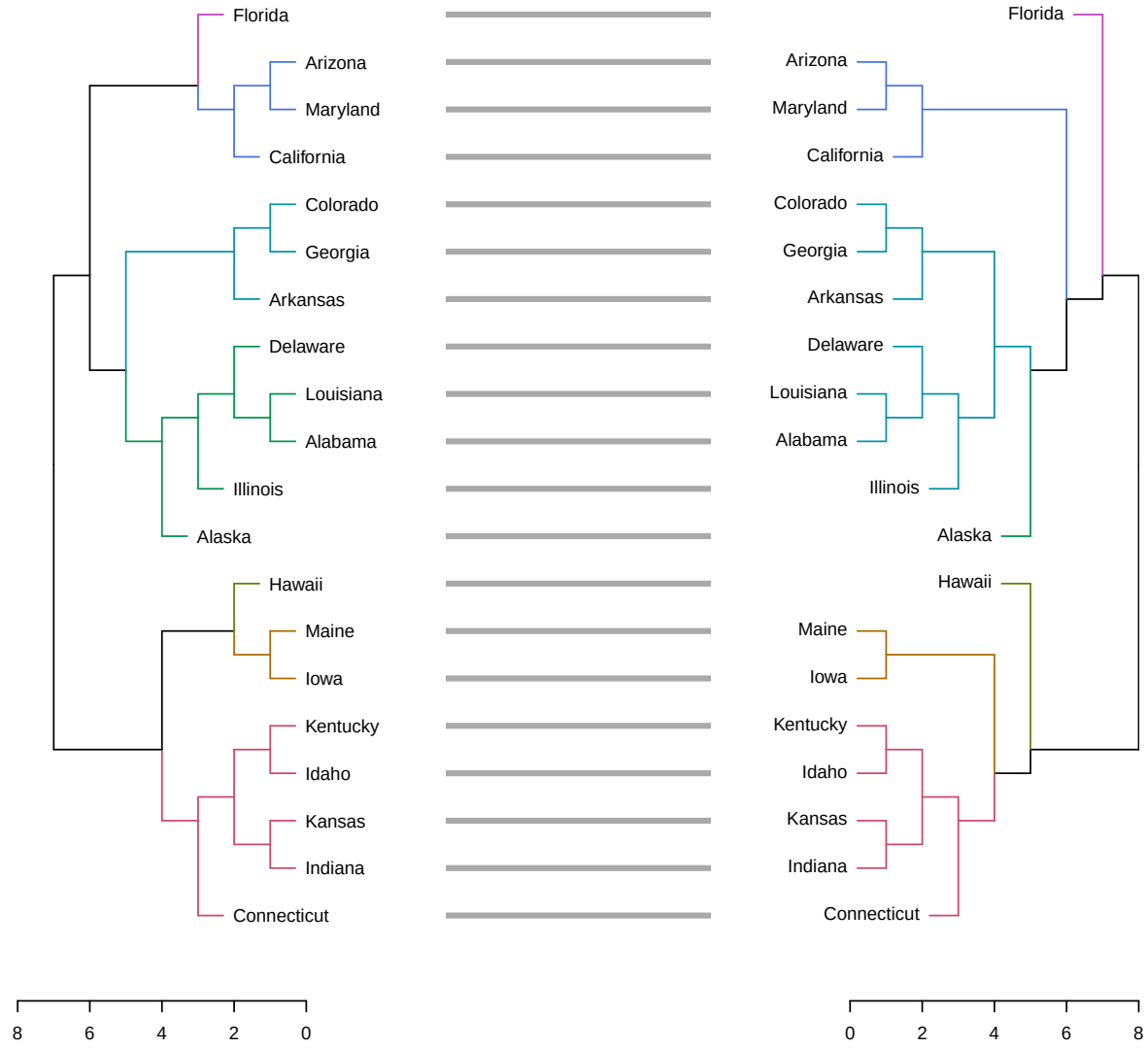

```
entanglement(dend12_corrected[[1]], dend12_corrected[[2]], L = 2)  # 0.0045

## [1] 0.004511

# best combination:
dend12_corrected_1 <- untangle_random_search(dend1, dend2)
dend12_corrected_2 <- untangle_step_rotate_2side(dend12_corrected_1[[1]],
    dend12_corrected_1[[2]])

## We ran untangle  1  times

tanglegram(dend12_corrected_2[[1]], dend12_corrected_2[[2]],
    margin_inner = 6.5)  # Better...
```




```
entanglement(dend12_corrected_2[[1]], dend12_corrected_2[[2]],
    L = 2)  # 0 - PERFECT.

## [1] 0

# using rank_branches and k_branches, combined with type,
# center, and xlim - helps to see various topological
# differences between the two trees:
tanglegram(dend12_corrected_2[[1]], dend12_corrected_2[[2]],
    k_branches = 7, rank_branches = TRUE, margin_inner = 6.5,
    type = "t", center = TRUE, xlim = c(8, 0))  # Much Better...
```




```
# using 'hang=TRUE' also helps:
tanglegram(dend12_corrected_2[[1]], dend12_corrected_2[[2]],
    k_branches = 7, rank_branches = TRUE, hang = TRUE, margin_inner = 6.5,
    type = "r", center = TRUE, xlim = c(8, 0))  # Also Better...
```



5. Comparing two trees - statistics and inference

5.1. Baker's Gamma Index

Baker's Gamma Index (Baker 1974) is a measure of accosiation (similarity) between two trees of heirarchical clustering (dendrograms). It is defined as the rank correlation between the stages at which pairs of objects combine in each of the two trees.

Or more detailed: It is calculated by taking two items, and see what is the heighst possible level of k (number of cluster groups created when cutting the tree) for which the two item still belongs to the same tree. That k is returned, and the same is done for these two items for the second tree. There are n over 2 combinations of such pairs of items from the items in the tree, and all of these numbers are calculated for each of the two trees. Then, these two sets of numbers (a set for the items in each tree) are paired according to the pairs of items compared, and a spearman correlation is calculated.

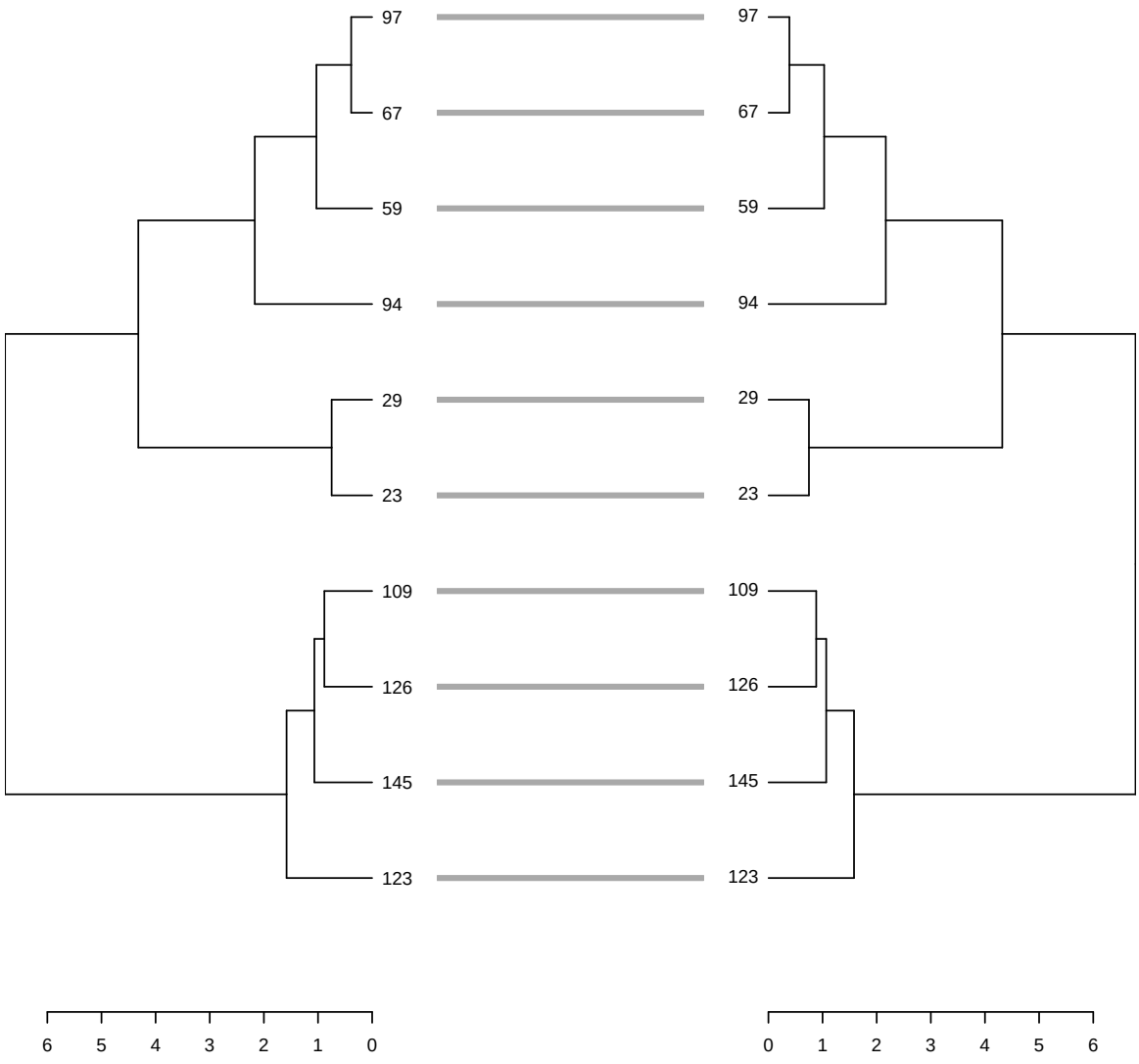
The value can range between -1 to 1. With near 0 values meaning that the two trees are not statistically similar. For exact p-value one should result to a permutation test. One such option will be to permute over the labels of one tree many times, and calculating the distriubtion under the null hypothesis (keeping the trees topologies constant).

Notice that this measure is not affected by the height of a branch but only of its relative position compared with other branches.

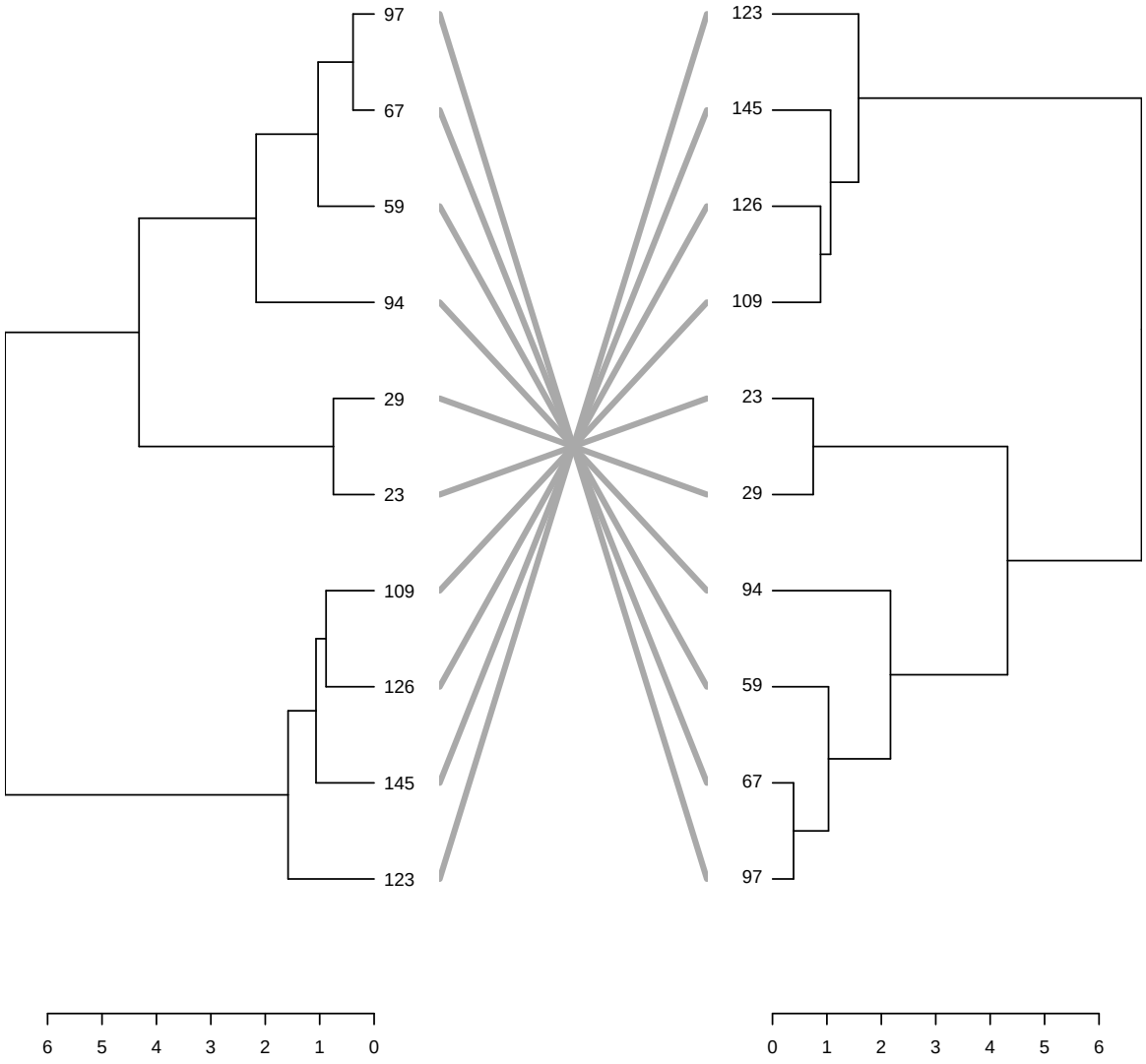
Here are a few examples:

```
set.seed(23235)
ss <- sample(1:150, 10)  # we want to compare small trees
hc1 <- hclust(dist(iris[ss, -5]), "com")
hc2 <- hclust(dist(iris[ss, -5]), "single")
dend1 <- as.dendrogram(hc1)
dend2 <- as.dendrogram(hc2)
# cutree(dend1)

cor_bakers_gamma(hc1, hc2)

## [1] 0.5716

cor_bakers_gamma(dend1, dend2)

## [1] 0.5716

# dend1 <- match_order_by_labels(dend1, dend2) # if you are
# not sure cor_bakers_gamma(dend1, dend2,
# use_labels_not_values = FALSE)

require(microbenchmark)
microbenchmark(with_labels = cor_bakers_gamma(dend1, dend2, try_cutree_hclust = FALSE),
    with_values = cor_bakers_gamma(dend1, dend2, use_labels_not_values = FALSE,
        try_cutree_hclust = FALSE), times = 10)

## Unit: milliseconds
##         expr   min     lq median     uq    max neval
##  with_labels 97.98 101.87 103.83 105.58 111.81    10
##  with_values 84.62  87.63  89.64  92.06  92.97    10

# The cor of a tree with itself is 1:
cor_bakers_gamma(dend1, dend1, use_labels_not_values = FALSE)
```

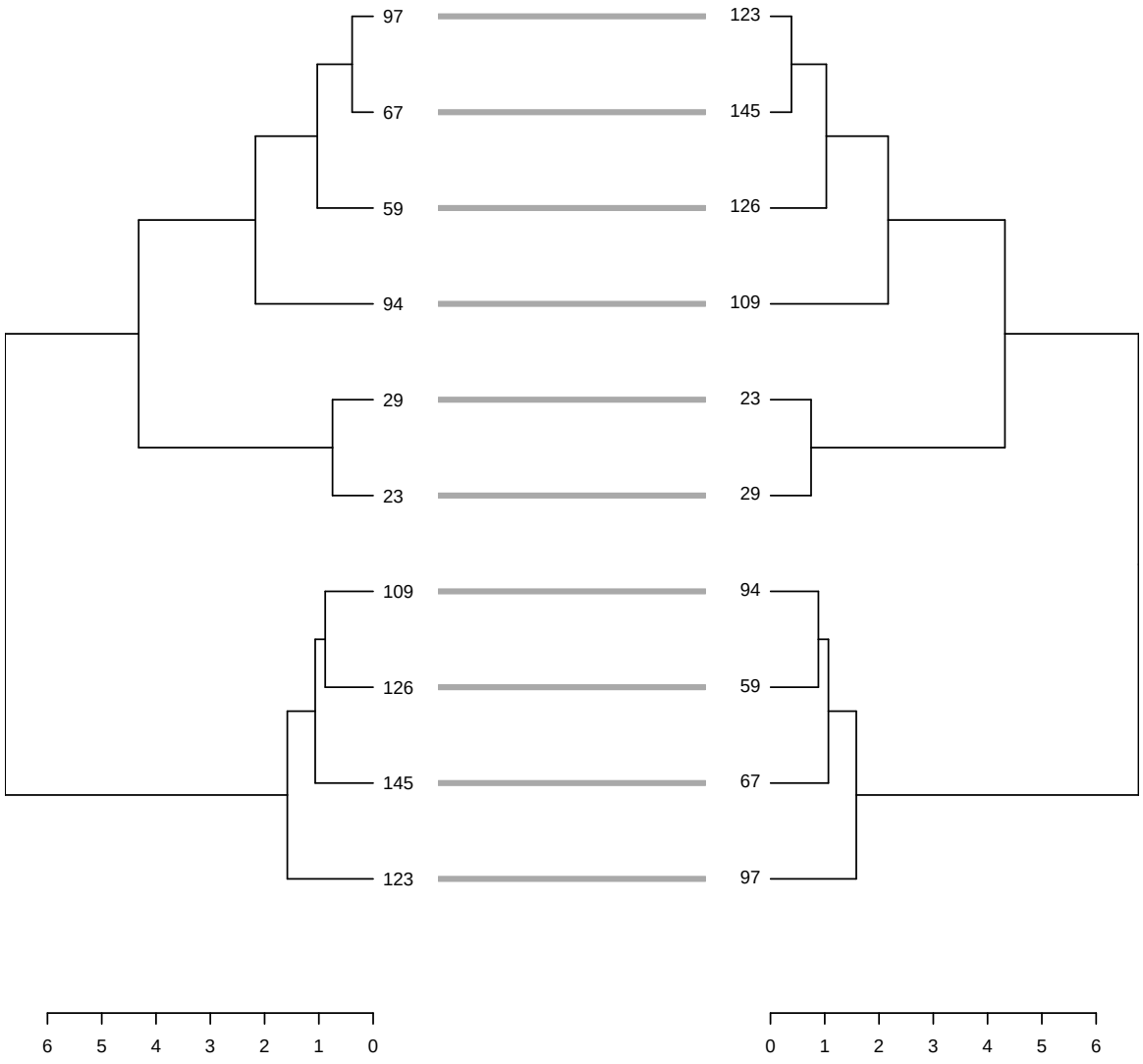
```
## [1] 1

cor_bakers_gamma(dend1, dend1, use_labels_not_values = TRUE)

## [1] 1

entanglement(dend1, dend1)  # having a worse entanglement

## [1] 0

tanglegram(dend1, dend1)  # having a worse entanglement
```



```
# tree order has no effect on the correlation:
rev_dend1 <- rev(dend1)
cor_bakers_gamma(dend1, rev_dend1, use_labels_not_values = TRUE)

## [1] 1

entanglement(dend1, rev_dend1)  # having a worse entanglement

## [1] 1

tanglegram(dend1, rev_dend1)  # having a worse entanglement
```

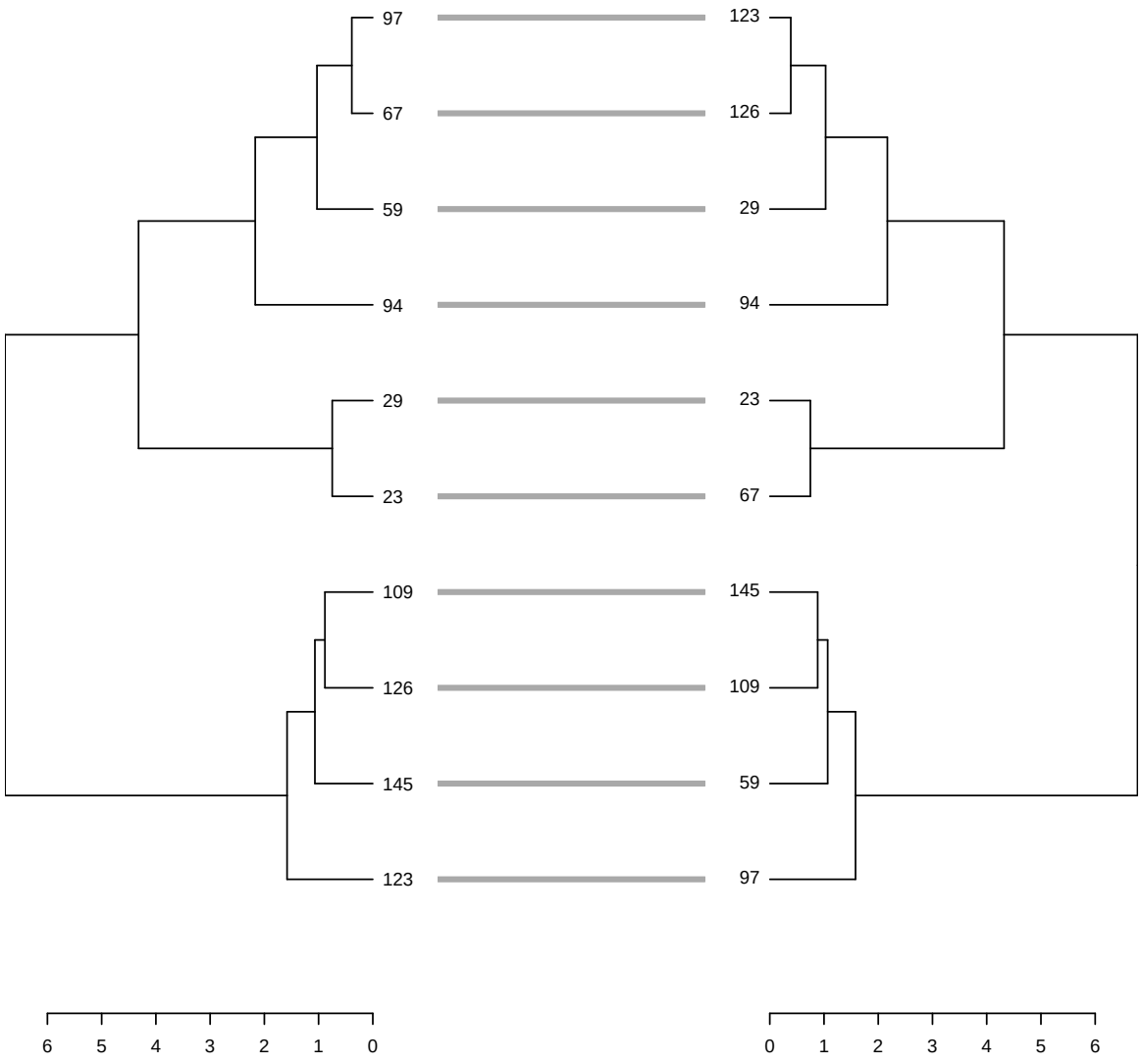


```
# But labels order does matter!!
dend1_mixed <- dend1
labels(dend1_mixed) <- rev(labels(dend1_mixed))
tanglegram(dend1, dend1_mixed)
```




```
entanglement(dend1, dend1_mixed)  # having the worst entanglement

## [1] 1

# does NOT mean having the worst cor!
cor_bakers_gamma(dend1, dend1_mixed, use_labels_not_values = TRUE)

## [1] 0.647
```

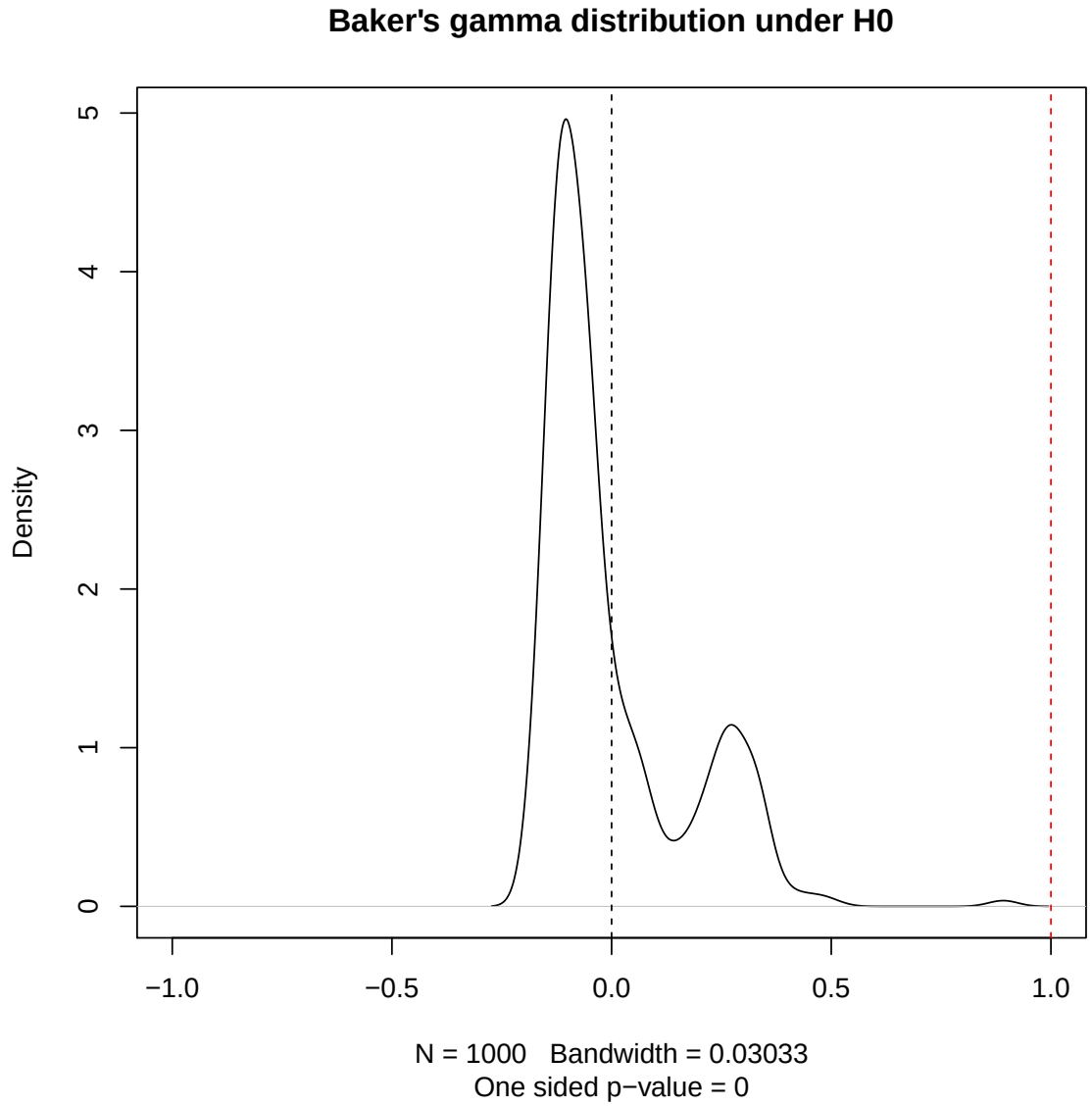
```
set.seed(983597)
labels(dend1_mixed) <- sample(labels(dend1_mixed))
tanglegram(dend1, dend1_mixed)
```




```
entanglement(dend1, dend1_mixed)  # having a worse entanglement

## [1] 0.7874

cor_bakers_gamma(dend1, dend1_mixed, use_labels_not_values = TRUE)

## [1] -0.06617
```

Since the observations creating the Baker's Gamma Index of such a measure are correlated, we need to perform a permutation test for the calculation of the statistical significance of the index. Let's look at the distribution of Baker's Gamma Index under the null hypothesis (assuming fixed tree topologies). This will be different for different tree structures and sizes. Here are the results when the compared tree is itself (after shuffaling its own labels), and when comparing tree 1 to the shuffled tree 2:

```
set.seed(23235)
ss <- sample(1:150, 10)  # we want to compare small trees
hc1 <- hclust(dist(iris[ss, -5]), "com")
hc2 <- hclust(dist(iris[ss, -5]), "single")
dend1 <- as.dendrogram(hc1)
dend2 <- as.dendrogram(hc2)
# cutree(dend1)

the_cor <- cor_bakers_gamma(dend1, dend1)
the_cor

## [1] 1

R <- 1000
cor_bakers_gamma_results <- numeric(R)
dend_mixed <- dend1
for (i in 1:R) {
    labels(dend_mixed) <- sample(labels(dend_mixed))
    cor_bakers_gamma_results[i] <- cor_bakers_gamma(dend1, dend_mixed)
}
plot(density(cor_bakers_gamma_results), main = "Baker's gamma distribution under H0",
    xlim = c(-1, 1))
abline(v = 0, lty = 2)
abline(v = the_cor, lty = 2, col = 2)
title(sub = paste("One sided p-value =", round(sum(the_cor <
    cor_bakers_gamma_results)/R, 4)))
```

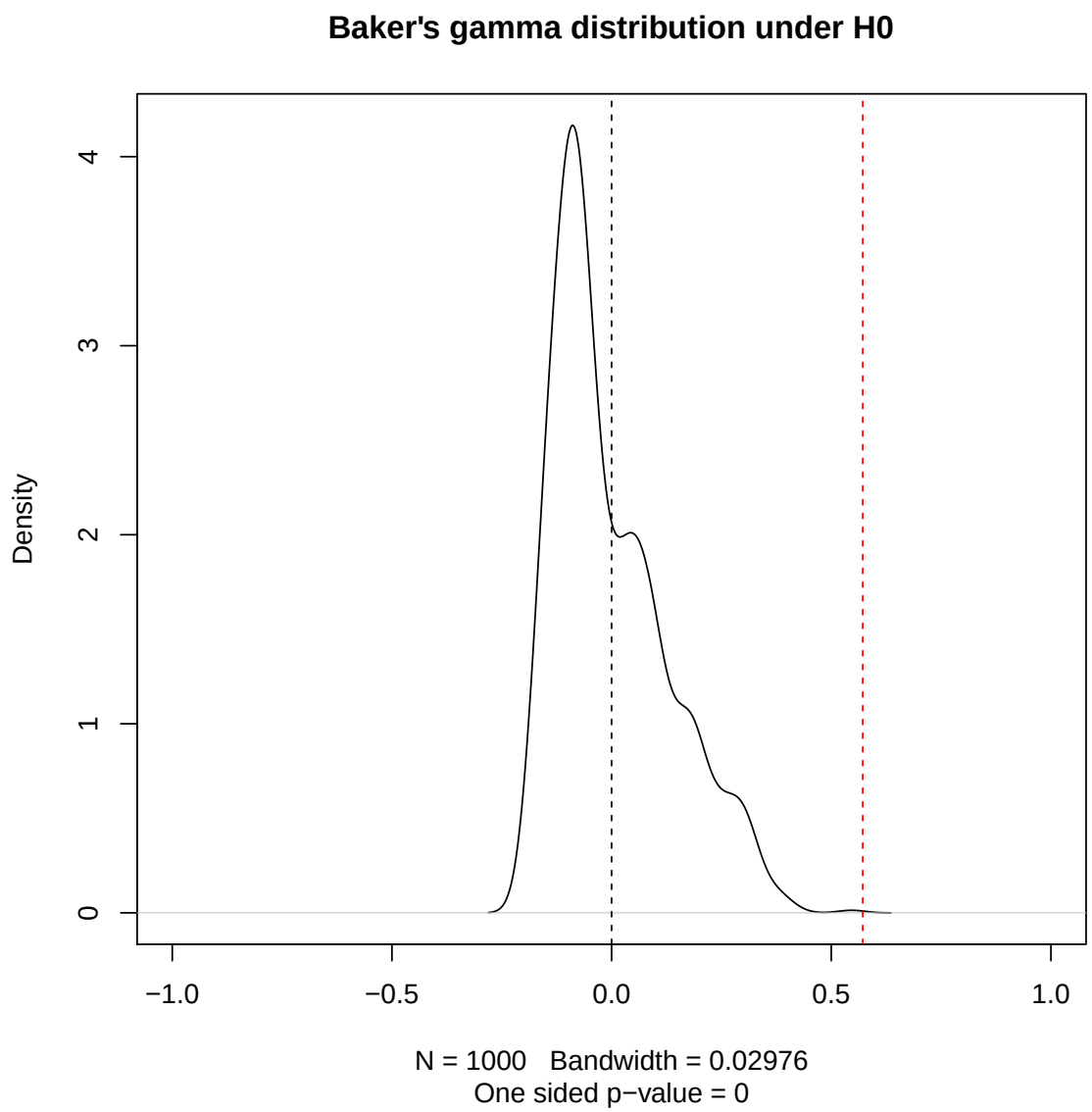



```
set.seed(23235)
ss <- sample(1:150, 10)  # we want to compare small trees
hc1 <- hclust(dist(iris[ss, -5]), "com")
hc2 <- hclust(dist(iris[ss, -5]), "single")
dend1 <- as.dendrogram(hc1)
dend2 <- as.dendrogram(hc2)
# cutree(dend1)

the_cor <- cor_bakers_gamma(dend1, dend2)
the_cor
```

```
## [1] 0.5716
```

```
R <- 1000
```

```
cor_bakers_gamma_results <- numeric(R)
dend_mixed <- dend2
for (i in 1:R) {
    labels(dend_mixed) <- sample(labels(dend_mixed))
    cor_bakers_gamma_results[i] <- cor_bakers_gamma(dend1, dend_mixed)
}
plot(density(cor_bakers_gamma_results), main = "Baker's gamma distribution under H0",
    xlim = c(-1, 1))
abline(v = 0, lty = 2)
abline(v = the_cor, lty = 2, col = 2)
title(sub = paste("One sided p-value =", round(sum(the_cor <
    cor_bakers_gamma_results)/R, 4)))
```



5.2. Bk method

6. Summary

The **dendextend** package presented in this paper greatly extends the available functionality of the dendrogram objects in R.

Acknowledgments

We are very thankful for code contributions and ideas by the R core team (especially Martin Maechler and Brian Ripley, but probably also others without our knowledge), Gavin Simpson, Gregory Jefferis ,

References

Baker FB (1974). "Stability of two hierarchical grouping techniques Case I: Sensitivity to data errors." *Journal of the American Statistical Association*, **69**(346), 440–445.

de Vries A, Ripley BD (2013). *ggdendro: Tools for extracting dendrogram and tree diagram plot data for use with ggplot.* R package version 0.1-12, URL http://CRAN.R-project.org/package=ggdendro.

Jefferis G (2013). *dendroextras: Extra functions to cut, label and colour dendrogram clusters.* R package version 0.1-2, URL http://CRAN.R-project.org/package=dendroextras.

Nia VP, Davison AC (2012). "High-Dimensional Bayesian Clustering with Variable Selection: The R Package bclust." *Journal of Statistical Software*, **47**(5), 1–22. URL http://www.jstatsoft.org/v47/i05/.

Nia VP, Stephens DA (2011). *Dendrogram Representation of Stochastic Clustering*, chapter 6, pp. 203–218. Nova Publishers. URL https://www.novapublishers.com/catalog/product_info.php?products_id=28702&osCsid=b.

Paradis E, Claude J, Strimmer K (2004). "APE: analyses of phylogenetics and evolution in R language." *Bioinformatics*, **20**, 289–290.

Ploner A (2012). *Heatplus: Heatmaps with row and/or column covariates and colored clusters.* R package version 2.6.0.

R Development Core Team (2013). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

Sarkar D, Andrews F (2012). *latticeExtra: Extra Graphical Utilities Based on Lattice.* R package version 0.6-24, URL http://CRAN.R-project.org/package=latticeExtra.

```
Sys.Date()

## [1] "2013-08-14"

sessionInfo()

## R version 3.0.1 (2013-05-16)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
##
## locale:
## [1] LC_COLLATE=Hebrew_Israel.1255
## [2] LC_CTYPE=Hebrew_Israel.1255
## [3] LC_MONETARY=Hebrew_Israel.1255
## [4] LC_NUMERIC=C
## [5] LC_TIME=Hebrew_Israel.1255
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets
## [6] methods   base
##
## other attached packages:
## [1] codetools_0.2-8  dendextend_0.7.3 ape_3.0-8
## [4] knitr_1.4
##
## loaded via a namespace (and not attached):
## [1] digest_0.6.3    evaluate_0.4.7  formatR_0.9
## [4] grid_3.0.1      lattice_0.20-15 nlme_3.1-109
## [7] stringr_0.6.2   tools_3.0.1
```

Affiliation:

Tal Galili
Department of Statistics and Operations Research
Tel Aviv University, Israel
E-mail: tal.galili@math.tau.ac.il
URLs: http://www.r-statistics.com/, http://www.r-bloggers.com/

Yoav Benjamini
The Nathan and Lily Silver
Professor of Applied Statistics
Department of Statistics and Operations Research
Tel Aviv University, Israel
E-mail: ybenja@tau.ac.il
URL: http://www.tau.ac.il/~ybenja/