RFLPtools: Analysis of DNA fragment samples and standalone BLAST report files

F. Flessa*, A. Kehl† and M. Kohl‡

October 23, 2012

Contents

1 Introduction

The package "RFLPtools" aims at

- the detection of similar band patterns based on DNA fingerprint fragment sizes (i.e. derived from RFLP-analysis)
- the analysis of standalone BLAST report files (i.e. DNA sequence analysis)

In this short vignette we describe and demonstrate the available functions.

```
> library(RFLPtools)
```

*Chair for Plant Systematics, Section Mycology, University of Bayreuth, 95440 Bayreuth, Germany, Fabienne.Flessa@uni-bayreuth.de

†Botanical Garden, University of Tübingen, Hartmeyerstr. 123, 72076 Tübingen, Germany, Alexandra.Kehl@botgarten.uni-tuebingen.de

‡Department of Mechanical and Process Engineering, Furtwangen University, 78054 VS-Schwenningen, Germany, Matthias.Kohl@stamats.de

2 RFLP data

We load example data and compute the Euclidean distance ...

```
> data(RFLPdata)
> res <- RFLPdist(RFLPdata)
> names(res) ## number of bands

 [1] "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12"

> str(res$"6")

Class 'dist'  atomic [1:210] 517.58 3.74 145.24 397.64 482.89 ...
  ..- attr(*, "Size")= int 21
  ..- attr(*, "Labels")= chr [1:21] "Ni_25_A3" "Ni_25_B1" "Ni_25_B3" "Ni_25_H5" ...
  ..- attr(*, "Diag")= logi FALSE
  ..- attr(*, "Upper")= logi FALSE
  ..- attr(*, "method")= chr "euclidean"
  ..- attr(*, "call")= language distfun(x = do.call("rbind", temp1))
```

Of course, we can also use other well-known distances implemented in function `dist`.

```
> res1 <- RFLPdist(RFLPdata, distfun = function(x) dist(x, method = "manhattan"))
> res2 <- RFLPdist(RFLPdata, distfun = function(x) dist(x, method = "maximum"))
> str(res[[1]])

Class 'dist'  atomic [1:28] 2.45 18.25 12.96 155.64 4.9 ...
  ..- attr(*, "Size")= int 8
  ..- attr(*, "Labels")= chr [1:8] "NI_28_D6" "Ni_28_A6" "Ni_28_B2" "Ni_28_C2" ...
  ..- attr(*, "Diag")= logi FALSE
  ..- attr(*, "Upper")= logi FALSE
  ..- attr(*, "method")= chr "euclidean"
  ..- attr(*, "call")= language distfun(x = do.call("rbind", temp1))

> str(res1[[1]])

Class 'dist'  atomic [1:28] 4 31 20 209 8 21 176 27 16 211 ...
  ..- attr(*, "Size")= int 8
  ..- attr(*, "Labels")= chr [1:8] "NI_28_D6" "Ni_28_A6" "Ni_28_B2" "Ni_28_C2" ...
  ..- attr(*, "Diag")= logi FALSE
  ..- attr(*, "Upper")= logi FALSE
  ..- attr(*, "method")= chr "manhattan"
  ..- attr(*, "call")= language dist(x = x, method = "manhattan")
```

```
> str(res2[[1]])

Class 'dist'  atomic [1:28] 2 13 10 146 4 8 104 12 9 147 ...
  ..- attr(*, "Size")= int 8
  ..- attr(*, "Labels")= chr [1:8] "NI_28_D6" "Ni_28_A6" "Ni_28_B2" "Ni_28_C2" ...
  ..- attr(*, "Diag")= logi FALSE
  ..- attr(*, "Upper")= logi FALSE
  ..- attr(*, "method")= chr "maximum"
  ..- attr(*, "call")= language dist(x = x, method = "maximum")
```

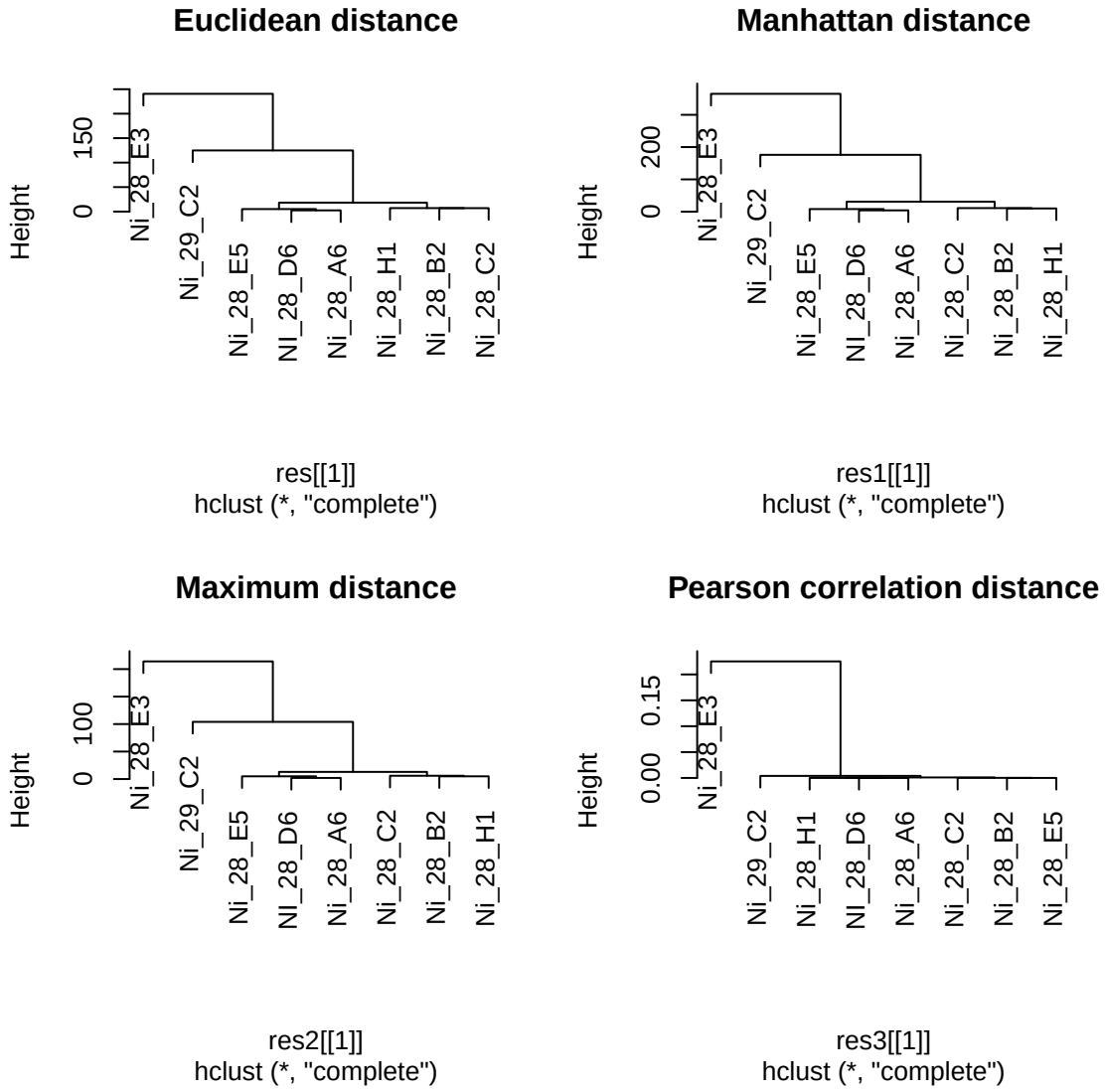
Correlation distances can be applied using function `corDist` of package `"MKmisc"`.

```
> library(MKmisc)
> res3 <- RFLPdist(RFLPdata, distfun = corDist)
> str(res3$"9")

Class 'dist'  atomic [1:21] 0.475 0.521 0.508 0.517 0.512 ...
  ..- attr(*, "Size")= int 7
  ..- attr(*, "Labels")= chr [1:7] "Ni_25_C4" "Ni_25_C5" "Ni_25_E4" "Ni_28_B9" ...
  ..- attr(*, "Diag")= logi FALSE
  ..- attr(*, "Upper")= logi FALSE
  ..- attr(*, "method")= chr "pearson"
  ..- attr(*, "call")= language distfun(x = do.call("rbind", temp1))
```

As we obtain a list of `dist` objects we can easily perform hierarchical clustering.

```
> par(mfrow = c(2,2))
> plot(hclust(res[[1]]), main = "Euclidean distance")
> plot(hclust(res1[[1]]), main = "Manhattan distance")
> plot(hclust(res2[[1]]), main = "Maximum distance")
> plot(hclust(res3[[1]]), main = "Pearson correlation distance")
```



For splitting the dendrogram into clusters we apply function `cutree`.

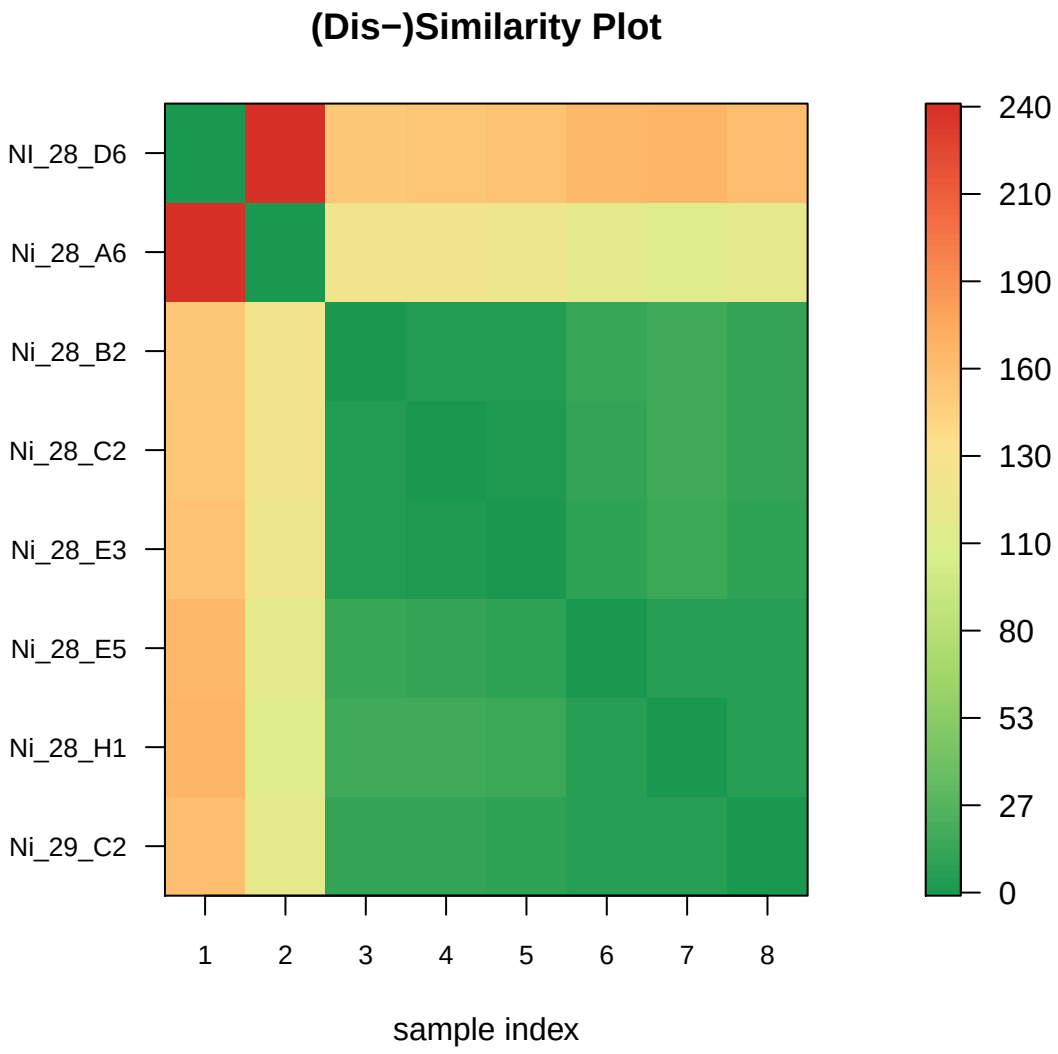
```
> clust4bd <- hclust(res[[2]])
> cgroups50 <- cutree(clust4bd, h=50)
> cgroups50

Ni_25_B2 Ni_25_B5 Ni_28_A2 Ni_28_A9 Ni_28_B4 Ni_28_B5 Ni_28_D4 Ni_28_E1
       1        2        3        4        5        3        3        6
Ni_28_E4 Ni_28_F2 Ni_28_F4 Ni_28_G4 Ni_28_H4 Ni_29_A3 Ni_29_A4 Ni_29_A7
```

```
       3        3        3        3        3        7        8        8
Ni_29_B4 Ni_29_B5 Ni_29_C7 Ni_29_D1 Ni_29_D6 Ni_29_D7 Ni_29_E4 Ni_29_E5
       9       10       11       12       13       14        8        9
Ni_29_F5 Ni_29_G1 Ni_29_G2 Ni_29_G4 Ni_29_H2 Ni_29_H4 Ni_29_H5
      11        7       11       15        7        8       13
```

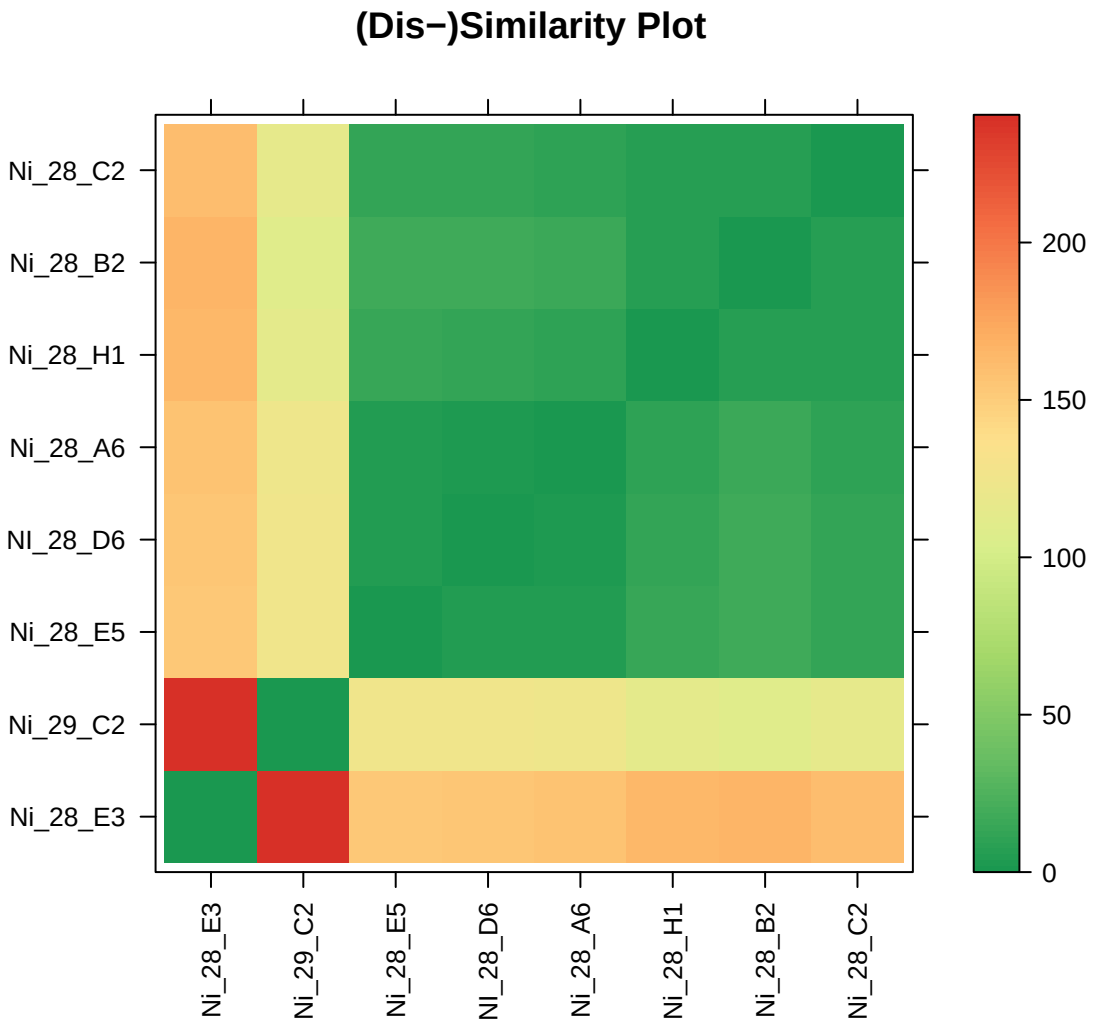
Another possibility to display the similarity of the samples are so-called (dis-)similarity matrices which can be generated by function `simPlot` of package `"MKmisc"`.

```
> library(RColorBrewer)
> library(MKmisc)
> myCol <- colorRampPalette(brewer.pal(8, "RdYlGn"))(128)
> ord <- order.dendrogram(as.dendrogram(hclust(res[[1]])))
> temp <- as.matrix(res[[1]])
> simPlot(temp[ord,ord], col = rev(myCol), minVal = 0,
+         labels = colnames(temp), title = "(Dis-)Similarity Plot")
```



We can also use function "levelplot" of "lattice" to display (dis-)similarity matrices.

```
> library(lattice)
> print(levelplot(temp[ord,ord], col.regions = rev(myCol),
+           at = do.breaks(c(0, max(temp)), 128),
+           xlab = "", ylab = "",
+           ## Rotate labels of x-axis
+           scales = list(x = list(rot = 90)),
+           main = "(Dis-)Similarity Plot"))
```



If some bands may be missing we can apply function `RFLPdist2` specifying the number of missing bands we expect.

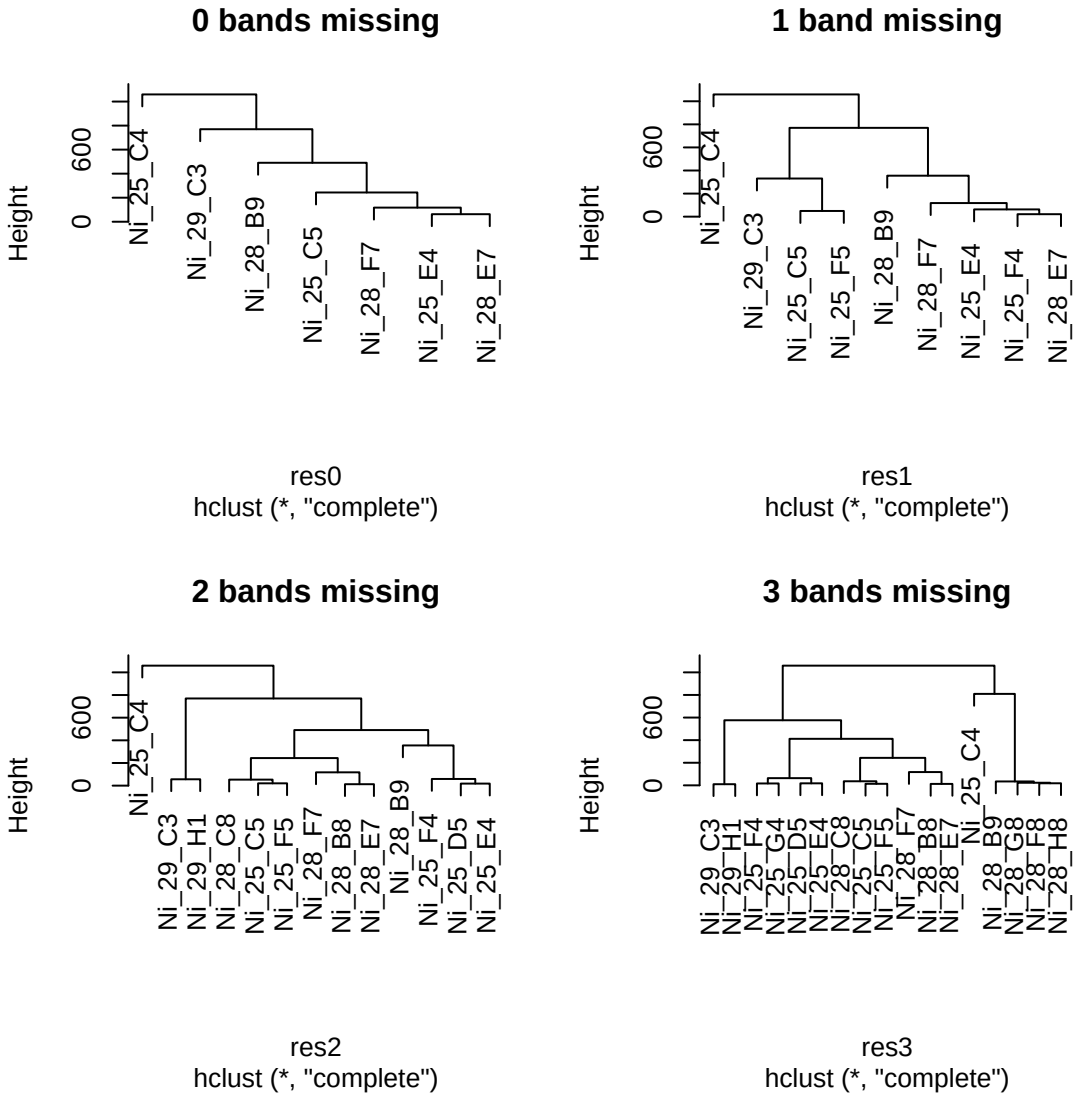
```
> ## Euclidean distance
> data(RFLPdata)
> data(RFLPref)
> nrBands(RFLPdata)

 [1]  3  4  5  6  7  8  9 10 11 12
```

```
> res0 <- RFLPdist2(RFLPdata, nrBands = 9, nrMissing = 0)
> res1 <- RFLPdist2(RFLPdata, nrBands = 9, nrMissing = 1)
> res2 <- RFLPdist2(RFLPdata, nrBands = 9, nrMissing = 2)
> res3 <- RFLPdist2(RFLPdata, nrBands = 9, nrMissing = 3)
```

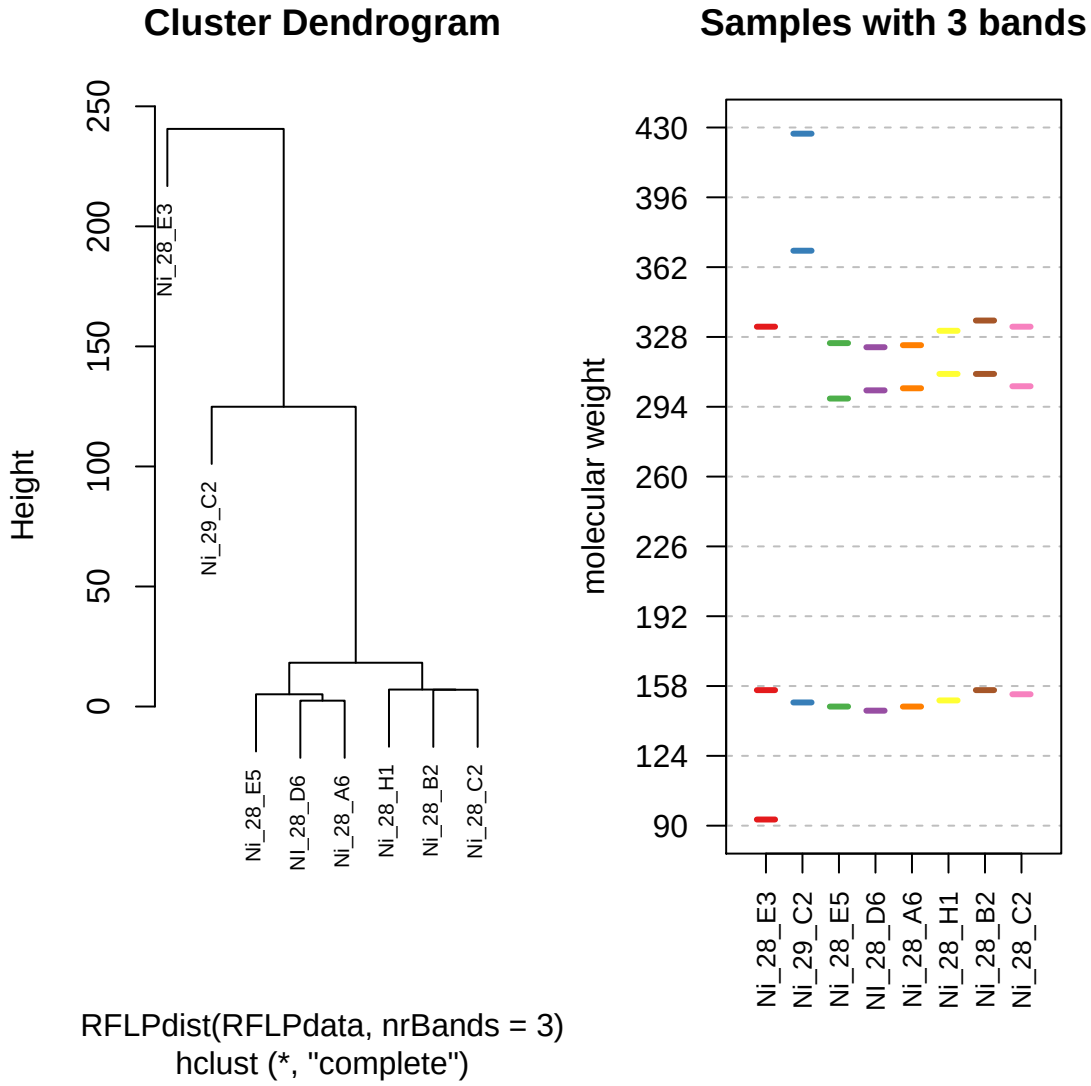
Again hierarchical clustering of the results is straight forward.

```
> par(mfrow = c(2,2))
> plot(hclust(res0), main = "0 bands missing")
> plot(hclust(res1), main = "1 band missing")
> plot(hclust(res2), main = "2 bands missing")
> plot(hclust(res3), main = "3 bands missing")
```



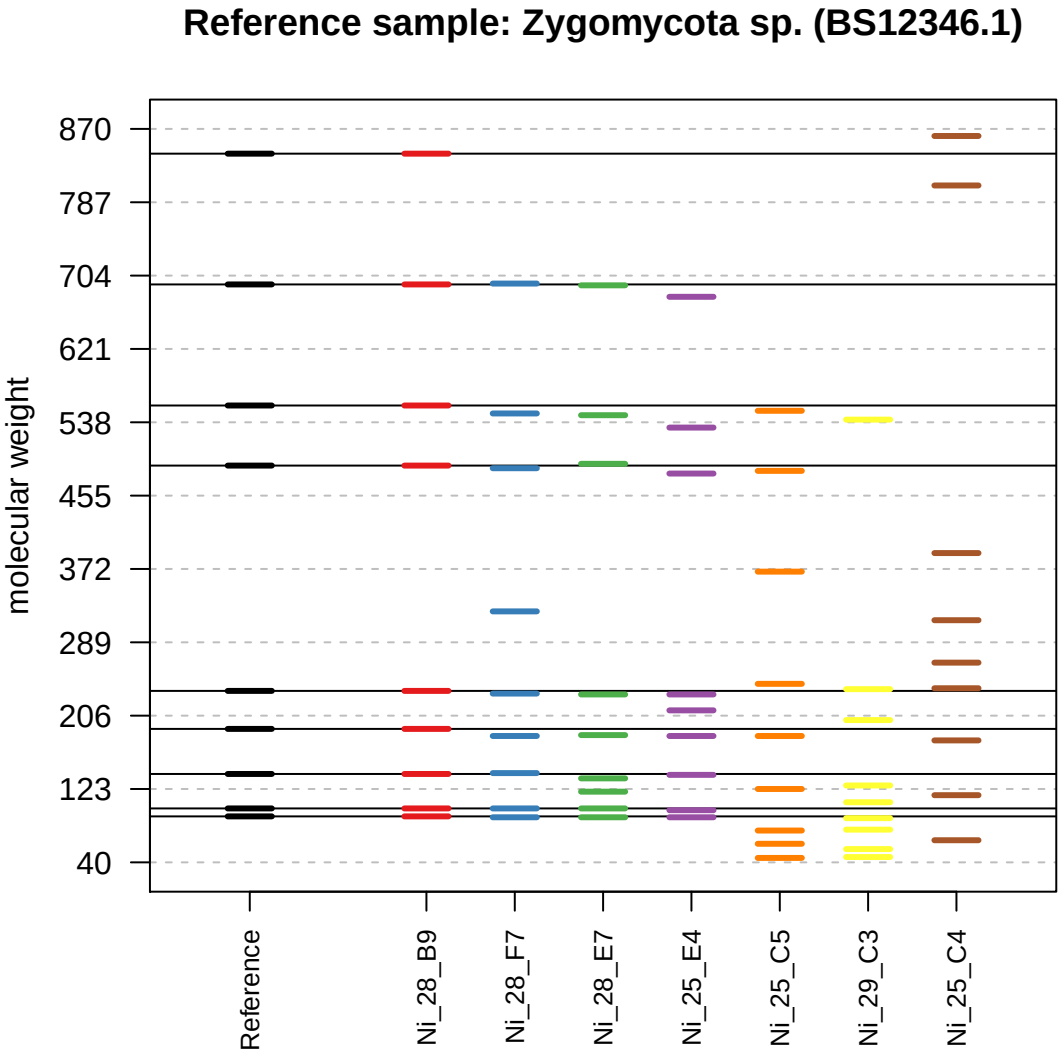
In function `RFLPplot` we have also implemented another possibility for visualization.

```
> par(mfrow = c(1,2))
> plot(hclust(RFLPdist(RFLPdata, nrBands = 3)), cex = 0.7)
> RFLPplot(RFLPdata, nrBands = 3, mar.bottom = 6, cex.axis = 0.8)
```



We can also make a comparison to reference data.

```
> RFLPrefplot(RFLPdata, RFLPref, nrBands = 9, cex.axis = 0.8)
```



3 BLAST data

To analyze tabular report files generated with standalone BLAST from NCBI (see `ftp://ftp.ncbi.nlm.nih.gov/blast/executables/release`), a function for reading the BLAST report files is included (`read.blast`). Possible steps are:

1) Install NCBI BLAST

2) Generate and import database(s)

3) Apply BLAST with options `outfmt` and `out`; e.g.

```
blastn -query Testquery -db Testdatabase -outfmt 6 -out out.txt
```

or

```
blastn -query Testquery -db Testdatabase -outfmt 10 -out out.csv
```

One could also call BLAST from inside R by using function `system`

```
system("blastn -query Testquery -db Testdatabase -outfmt 6 -out out.txt")
```

4) Read in the results

```
## -outfmt 6
test.res <- read.blast(file = "out.txt")
```

or

```
## -outfmt 10
test.res <- read.blast(file = "out.csv", sep = ",")
```

We now read in a example file included in folder `extdata` of our package.

```
> Dir <- system.file("extdata", package = "RFLPtools") # input directory
> filename <- file.path(Dir, "BLASTexample.txt")
> BLAST1 <- read.blast(file = filename)
> str(BLAST1)

'data.frame':        4069 obs. of  12 variables:
 $ query.id        : chr  "agrFF002" "agrFF002" "agrFF002" "agrFF002" ...
 $ subject.id      : chr  "agrFF002" "agrFF148" "agrFF148" "agrFF176" ...
 $ identity        : num  100 93.4 100 91.4 100 ...
 $ alignment.length: int  544 243 11 255 11 255 11 256 11 256 ...
 $ mismatches      : int  0 14 0 20 0 20 0 18 0 18 ...
 $ gap.opens       : int  0 2 0 2 0 2 0 3 0 3 ...
 $ q.start         : int  1 199 462 187 462 187 462 187 462 187 ...
 $ q.end           : int  544 439 472 439 472 439 472 439 472 439 ...
 $ s.start         : int  1 671 785 123 250 121 248 121 248 126 ...
 $ s.end           : int  544 913 795 377 260 375 258 375 258 380 ...
 $ evalue          : num  0.0 6.0e-102 6.7 2.0e-100 6.7 ...
 $ bit.score       : num  944 360 21.1 354 21.1 354 21.1 352 21.1 352 ...
```

This example BLAST data is also available as loadable example data.

```
> data(BLASTdata)
```

The loaded `data.frame` can be used to compute similarities between the BLASTed sequences via function `simMatrix`. This function includes the following steps:

1. the length of each sequence (`LS`) comprised in the input data file is extracted.
2. if there is more than one comparison for one sequence including different parts of the respective sequence, that one with maximum base length is chosen.
3. the number of matching bases (`mB`) is calculated by multiplying two variables given in the BLAST output: the identity between sequences (%) and the number of nucleotides divided by 100.
4. the resulting value is rounded to the next integer.
5. the similarity is calculated by dividing `mB` by `LS` and saved in the corresponding similarity matrix.

If the similarity of a combination is not shown in the BLAST report file (because the similarity was lower than 70%), this comparison is included in the similarity matrix with the result zero.

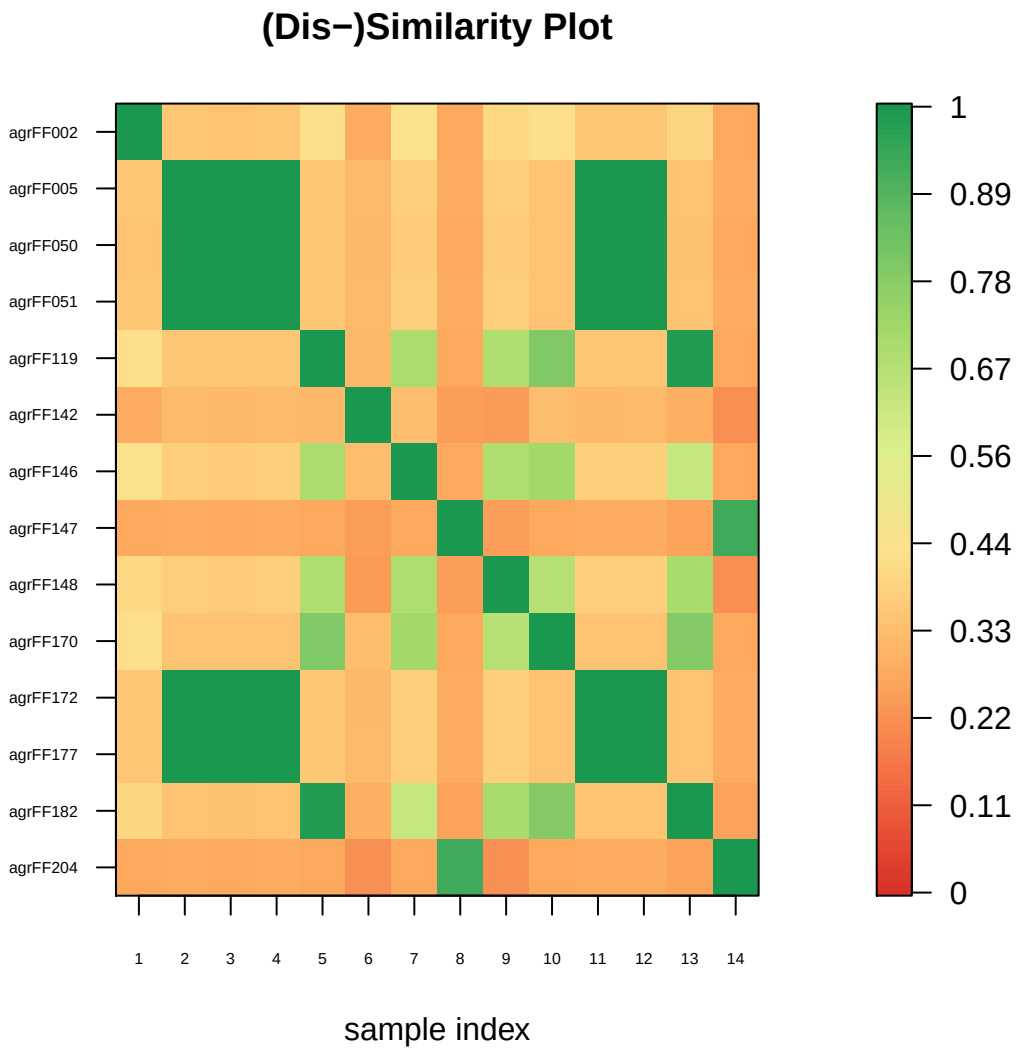
```
> res <- simMatrix(BLASTdata)
```

Optionally, the range of sequence length can be specified to exclude sequences which were too short or too long, respectively.

```
> res1 <- simMatrix(BLASTdata, sequence.range = TRUE, Min = 100, Max = 450)
> res2 <- simMatrix(BLASTdata, sequence.range = TRUE, Min = 500)
```

We display the similarity matrix.

```
> library(MKmisc)
> simPlot(res2, col = myCol, minVal = 0, cex.axis = 0.5,
+         labels = colnames(res2), title = "(Dis-)Similarity Plot")
```

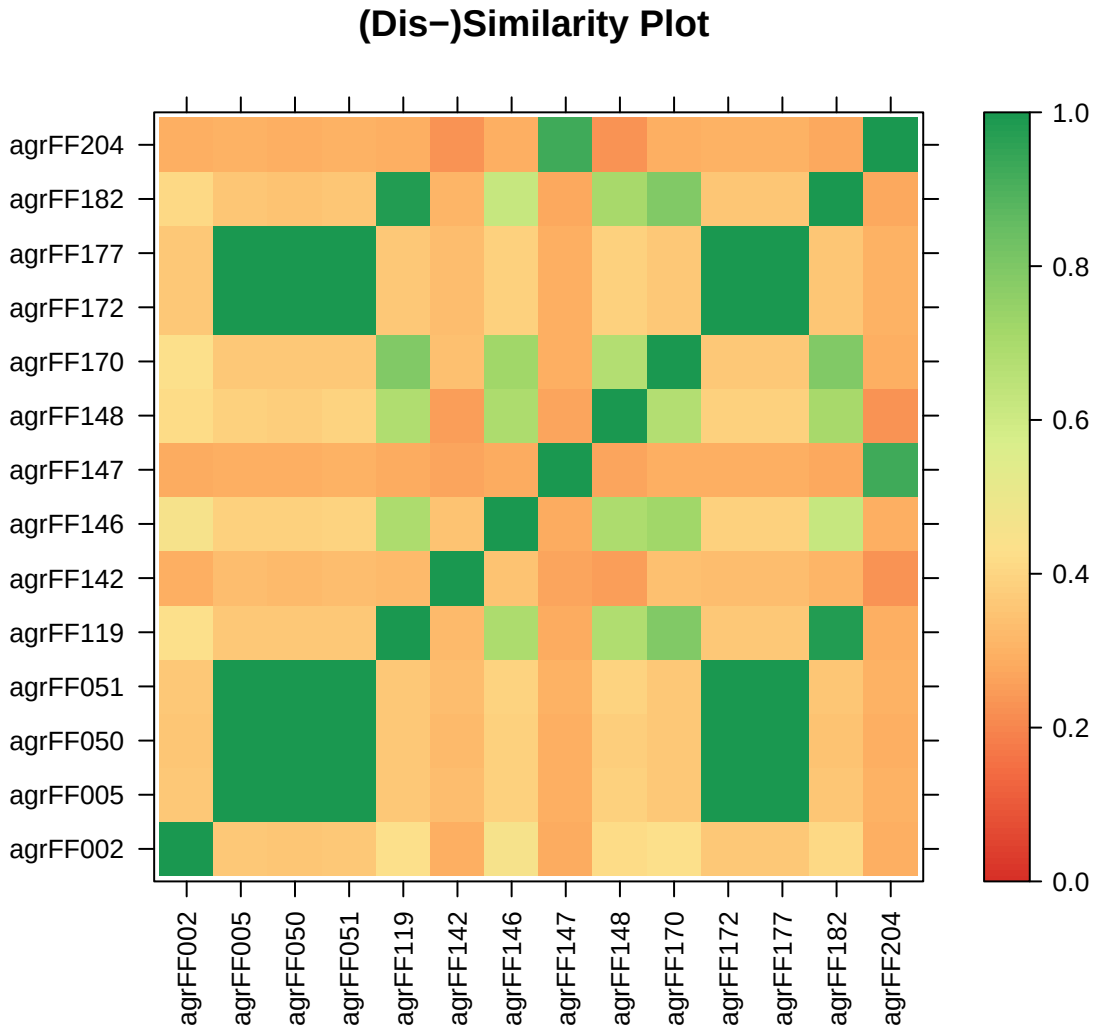


Alternatively, we can again use function `levelplot` of package `"lattice"`.

```
> library(lattice)
> txt <- trellis.par.get("add.text")
> txt$cex <- 0.5
> trellis.par.set("add.text" = txt)
> myCol <- colorRampPalette(brewer.pal(8, "RdYlGn"))(128)

> print(levelplot(res2, col.regions = myCol,
+           at = do.breaks(c(0, max(res2)), 128),
```

```
+            xlab = "", ylab = "",
+            ## Rotate labels of x axis
+            scales = list(x = list(rot = 90)),
+            main = "(Dis-)Similarity Plot"))
```

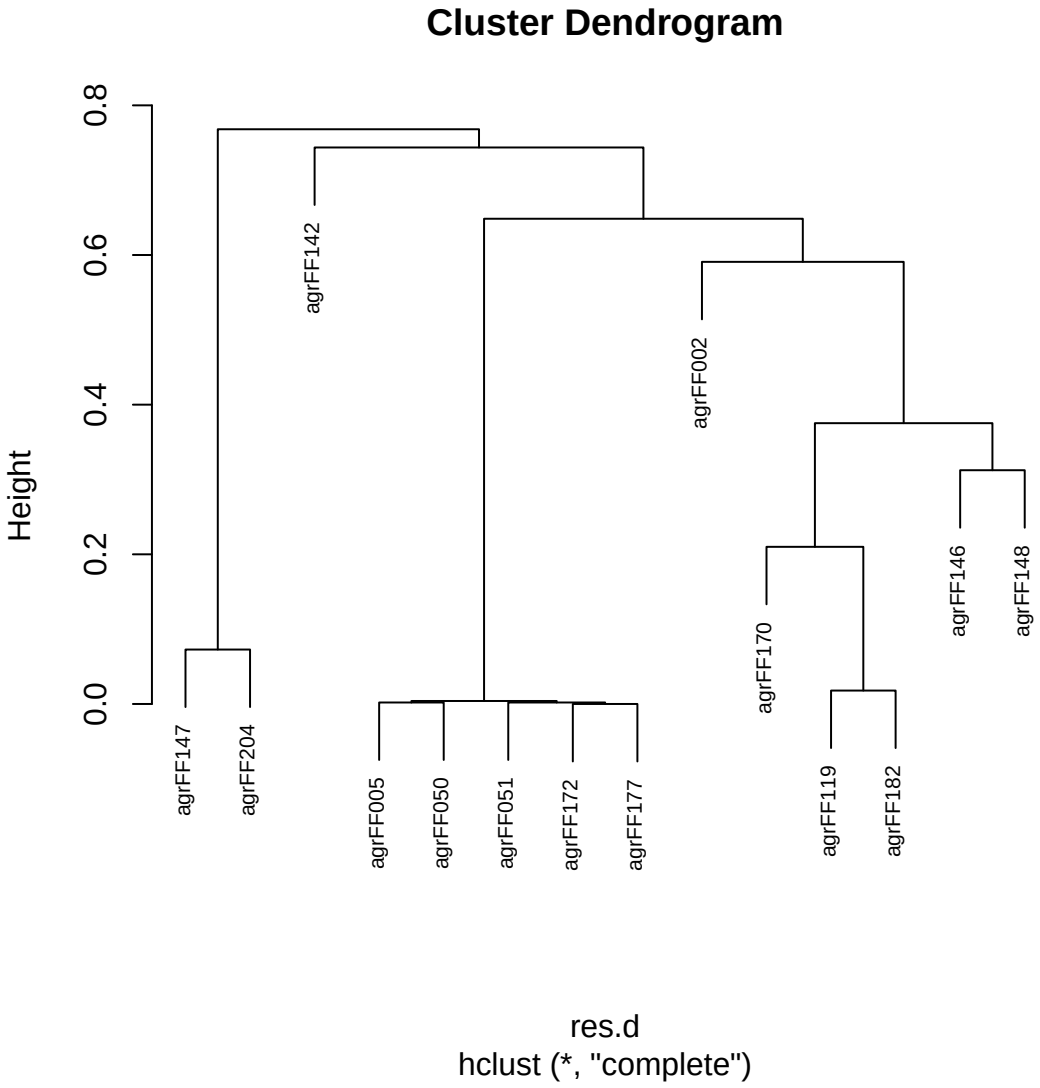


We can also convert the similarity matrix to an object of S3 class `"dist"`.

```
> res.d <- sim2dist(res2)
```

After the conversion we can for instance perform hierarchical clustering.

```
> ## hierarchical clustering
> plot(hclust(res.d), cex = 0.7)
```



References

[1] Poussier, Stephane; Trigalet-Demery, Danielle; Vandewalle, Peggy; Goffinet, Bruno; Luisetti, Jacques; Trigalet, Andre. Genetic diversity of Ralstonia solanacearum as

assessed by PCR-RFLP of the hrp gene region, AFLP and 16S rRNA sequence analysis, and identification of an African subdivision. Microbiology 2000 146:1679-1692

[2] Matsumoto, Masaru; Furuya, Naruto; Takanami, Yoichi; Matsuyama, Nobuaki. RFLP analysis of the PCR-amplified 28S rDNA in Rhizoctonia solani. Mycoscience 1996 37:351 - 356