RedeR: bridging the gap between hierarchical network representation and functional analysis.

Mauro AA Castro, Xin Wang, Michael NC Fletcher, Kerstin B Meyer and Florian Markowetz.

30 October 2017

Abstract

RedeR is an R-based package combined with a stand-alone Java application for interactive visualization and manipulation of modular structures, nested networks and multiple levels of hierarchical associations. Supporting information is available at Castro et al. (2016) (http://genomebiology.com/2012/13/4/R29).

Package

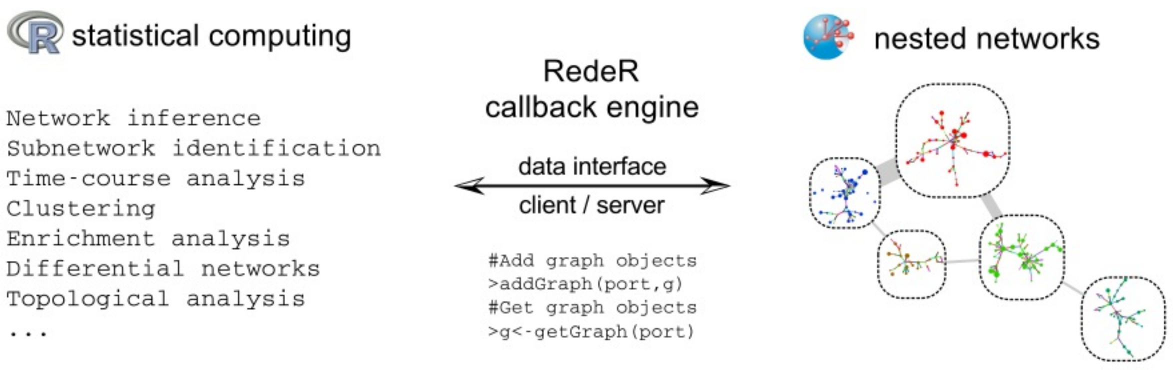
RedeR 1.26.0

Contents

1 Overview

RedeR is an R-based package combined with a Java application for dynamic network visualization and manipulation. It implements a callback engine by using a low-level R-to-Java interface to run complementary methods. In this sense, **RedeR** takes advantage of **R** to run robust statistics, while the R-to-Java interface bridge the gap between network analysis and visualization.

The package is designed to deal with three key challenges in network analysis. Firstly, biological networks are modular and hierarchical, so network visualization needs to take advantage of such structural features. Secondly, network analysis relies on statistical methods, many of which are already available in resources like CRAN or Bioconductor. However, the missing link between advanced visualization and statistical computing makes it hard to take full advantage of R packages for network analysis. Thirdly, in larger networks user input is needed to focus the view of the network on the biologically relevant parts, rather than relying on an automatic layout function (additional information is available at Castro et al. (2012)). The design of the software is depicted from **Figure 1**. Complex graphs with many attributes can be transferred from-and-to **R** using `addGraph` and `getGraph` functions.



```
#Add graph objects
>addGraph(port,g)
#Get graph objects
>g<-getGraph(port)
```

Figure 1. Schematic representation of RedeR calls. In the low-level interface, the Apache server ("The Apache Software Foundation" 2010) is used to link R to Java.

2 Quick Start

2.1 Main callback methods

The first step is to build the server port, which will be required in all remote procedure calls. By default the constructor `RedPort` should set all details:

```
library(RedeR)
```

```
## ***This is RedeR 1.26.0! For a quick start, please type 'vignette('RedeR')'.
##    Supporting information is available at Genome Biology 13:R29, 2012,
##    (doi:10.1186/gb-2012-13-4-r29).
```

```
rdp <- RedPort()
```

Next, invoke RedeR using the method `calld`:

```
calld(rdp)
```

Within an active interface, then the method `addGraph` can easily send R graphs to the application. For example, the following chunk adds an **igraph** object (**Fig. 2**):

```
library (igraph)
g1 <- graph.lattice(c(5,5,5))
addGraph( rdp, g1, layout.kamada.kawai(g1) )
```

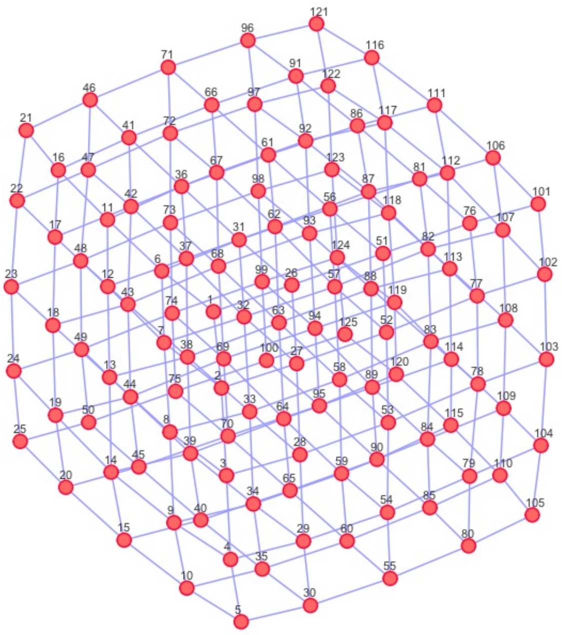


Figure 2. A toy example added to **RedeR** by the `addGraph` function.

Conversely, RedeR graphs can be transferred to R and wrapped in **igraph** objects:

```
g2 <- getGraph(rdp)
resetd(rdp)
```

The interface accepts additional graph attributes, as for example edge direction, edge width, edge weight, node shape, node size, node color etc. In order to pass these extensible features the attributes must be provided in a valid syntax (see `getGraph` and `addGraph` specification for additional details).

Another strategy is to wrap graphs into containers and then send it to the Java application. Next, the subgraphs `g3`and `g4` are assigned to different nested structures (**Fig. 3**).

```
g3 <- barabasi.game(10)
g4 <- barabasi.game(10)
V(g3)$name<-paste("sn",1:10,sep="")
V(g4)$name<-paste("sm",1:10,sep="")
addGraph(rdp, g3, isNest =TRUE, gcoord=c(25,25), gscale=50)
addGraph(rdp, g4, isNest =TRUE, gcoord=c(75,75), gscale=50)
```

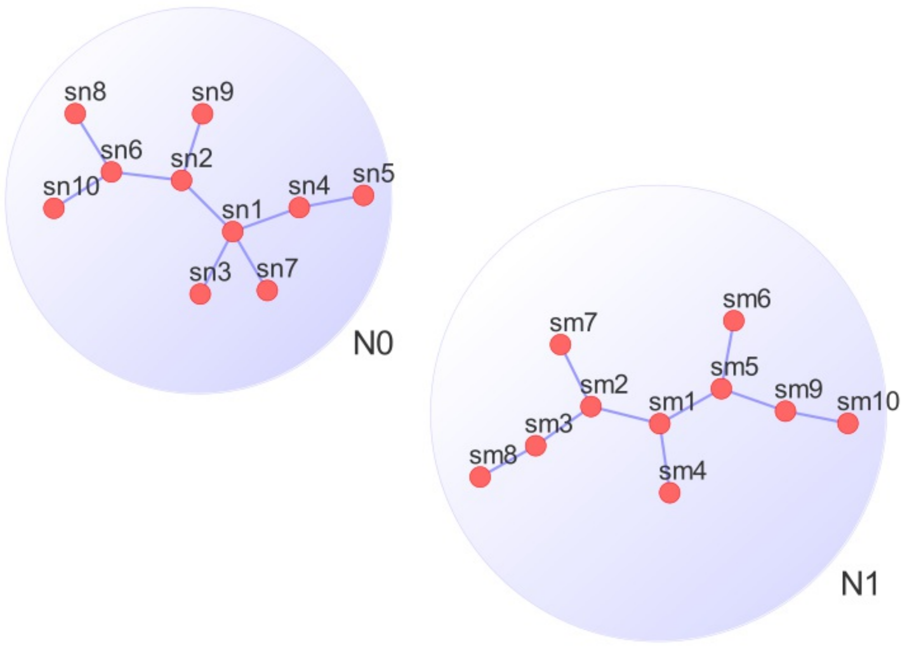


Figure 3. Nested graphs in **RedeR** using the command `addGraph`.

In this case, the subgraphs can be handled apart from each other. For example, the following chunk selects all nodes assigned to the container “N0” and then gets back the subgraph (the selection step can also be done interactively).

```
selectNodes(rdp,"N0")
g5 <- getGraph(rdp, status= "selected")
resetd(rdp)
```

As a suggestion, try some RedeR features in the Java side (e.g. open samples s2 or s3 in the main panel and enjoy the dynamic layout options!).

2.2 Interactive work

The next chunk generates a scale-free graph according to the Barabasi-Albert model (Csardi and Nepusz 2006) and sends the graph to **RedeR** without any layout information.

```
g6 <- barabasi.game(500)
addGraph(rdp, g6, zoom=20)
```

Then using the “relax” options available in the app you can tune the graph layout as presented in **Figure 4**.

```
relax(rdp,p2=400,p5=30,ps=T)
```

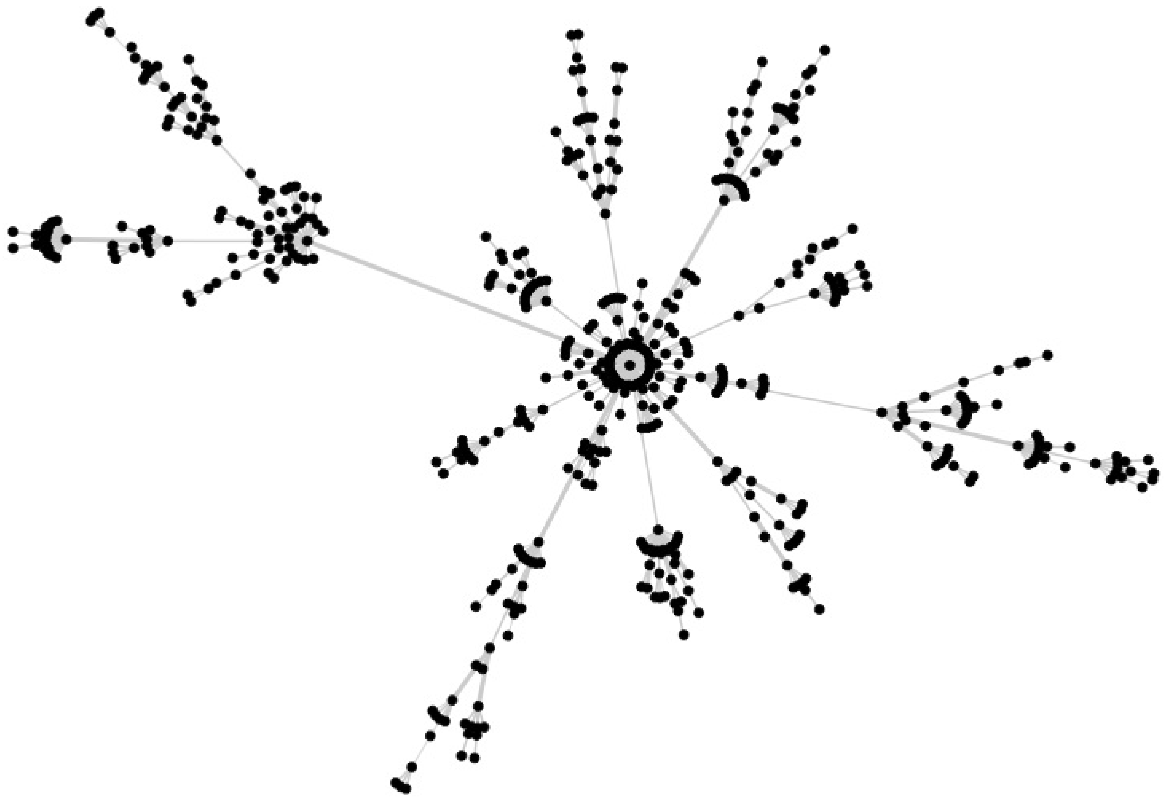


Figure 4. Scale-free graph according to the Barabasi-Albert model (Csardi and Nepusz 2006).

In **Figure 5** the same graph is used to exemplify the community structure mapped by the edge-betweenness algorithm (available on both the Java interface and the ** igraph ** package). In **Figure 6** these communities are nested to hidden containers, which are objects of the same class of nodes but with additional behaviors. You can build these containers either using **R** or **Java** functions (see options available in the **clustering** main menu and in the shortcuts of the nested objects, i.e., right-click a container).

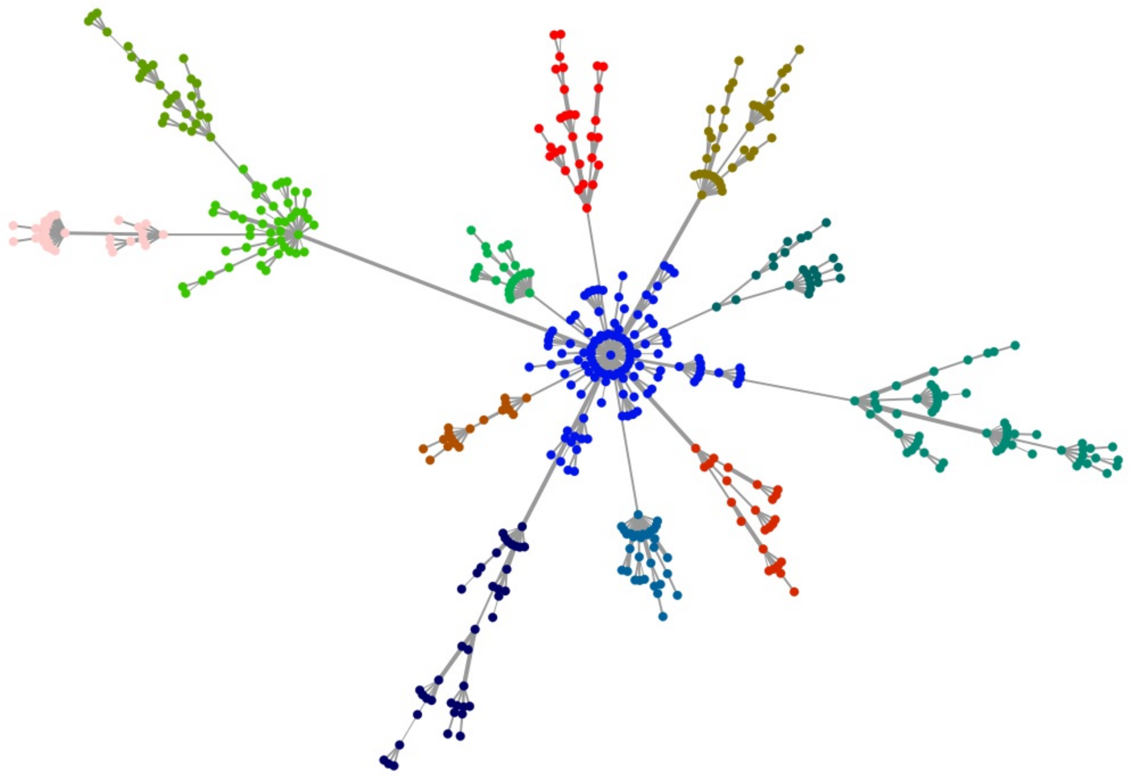


Figure 5. Subgraphs detected based on edge betweenness.

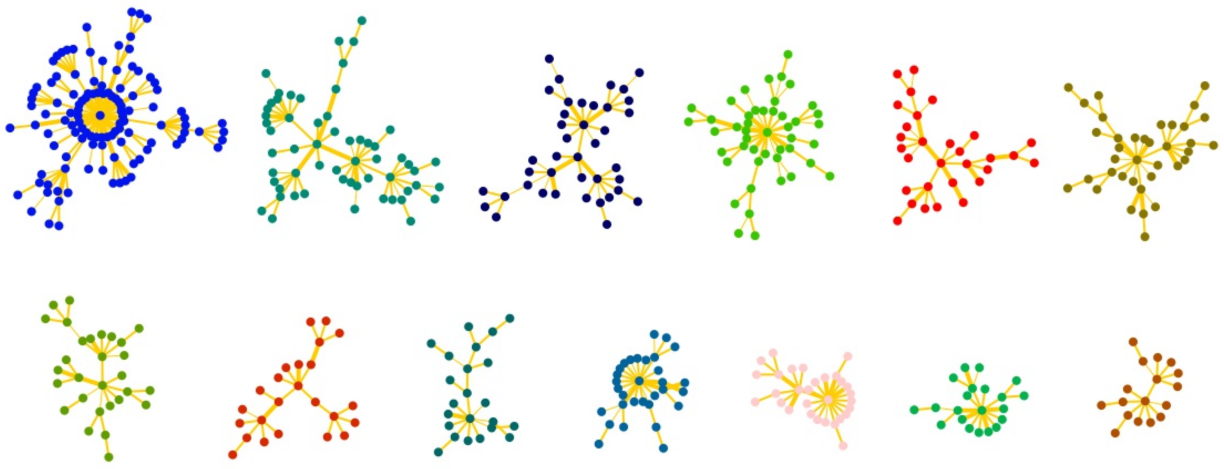


Figure 6. Subnetworks within hidden containers.

3 Workflow illustration

This section provides a sequence of steps that illustrates how users might integrate its own pre-processed data in a given network to visualize subgraphs and nested networks. Please refer to Castro et al. (2012) for more details about the biological background and experimental design of each example.

3.1 Subgraphs

Start the app (i.e. run the ‘calld’ method), and then get the indicated data frame and interactome:

```
data(ER.limma)
data(hs.inter)
dt <- ER.limma
gi <- hs.inter
```

Extract and set attributes to a subgraph represented by genes differentially expressed at a given time point (i.e. logFC from t3-t0 contrast):

```
gt3  <- subg(g=gi, dat=dt[dt$degenes.t3!=0,], refcol=1)
```

```
## ...note: not all genes found in the network!
```

```
gt3  <- att.setv(g=gt3, from="Symbol", to="nodeAlias")
gt3  <- att.setv(g=gt3, from="logFC.t3", to="nodeColor", breaks=seq(-2,2,0.4), pal=2)
```

Note that some genes will not be found in the interactome, and that’s okay. Extract another subgraph and set its attributes (i.e. logFC from t6-t0 contrast):

```
gt6  <- subg(g=gi, dat=dt[dt$degenes.t6!=0,], refcol=1)
```

```
## ...note: not all genes found in the network!
```

```
gt6  <- att.setv(g=gt6, from="Symbol", to="nodeAlias")
gt6  <- att.setv(g=gt6, from="logFC.t6", to="nodeColor", breaks=seq(-2,2,0.4), pal=2)
```

Extract another subgraph and set its attributes (i.e. logFC from t12-t0 contrast):

```
gt12 <- subg(g=gi, dat=dt[dt$degenes.t12!=0,], refcol=1)
```

```
## ...note: not all genes found in the network!
```

```
gt12 <- att.setv(g=gt12, from="Symbol", to="nodeAlias")
gt12 <- att.setv(g=gt12, from="logFC.t12", to="nodeColor", breaks=seq(-2,2,0.4), pal=2)
```

Now add all subgraphs to the app (**Fig.7**):

```
addGraph(rdp, gt3, gcoord=c(10,25), gscale=20, isNest=TRUE, theme='tm1', zoom=30)
addGraph(rdp, gt6, gcoord=c(20,70), gscale=50, isNest=TRUE, theme='tm1', zoom=30)
addGraph(rdp, gt12, gcoord=c(70,55), gscale=80, isNest=TRUE, theme='tm1', zoom=30)
```

... and nest overlapping subgraphs (i.e. indicating overlapping time series!):

```
nestNodes(rdp, nodes=V(gt3)$name, parent="N1", theme='tm2')
nestNodes(rdp, nodes=V(gt6)$name, parent="N2", theme='tm2')
nestNodes(rdp, nodes=V(gt3)$name, parent="N4", theme='tm3')
```

To simplify the graph, the `mergeOutEdges` method assigns edges to containers:

```
mergeOutEdges(rdp)
```

Relax the network:

```
relax(rdp,50,400)
```

Add color legend (other types are available):

```
scl <- gt3$legNodeColor$scale
leg <- gt3$legNodeColor$legend
addLegend.color(rdp, colvec=scl, labvec=leg, title="node color (logFC)")
```

Select a gene:

```
selectNodes(rdp,"RET")
```

Reset graph:

```
resetd(rdp)
```

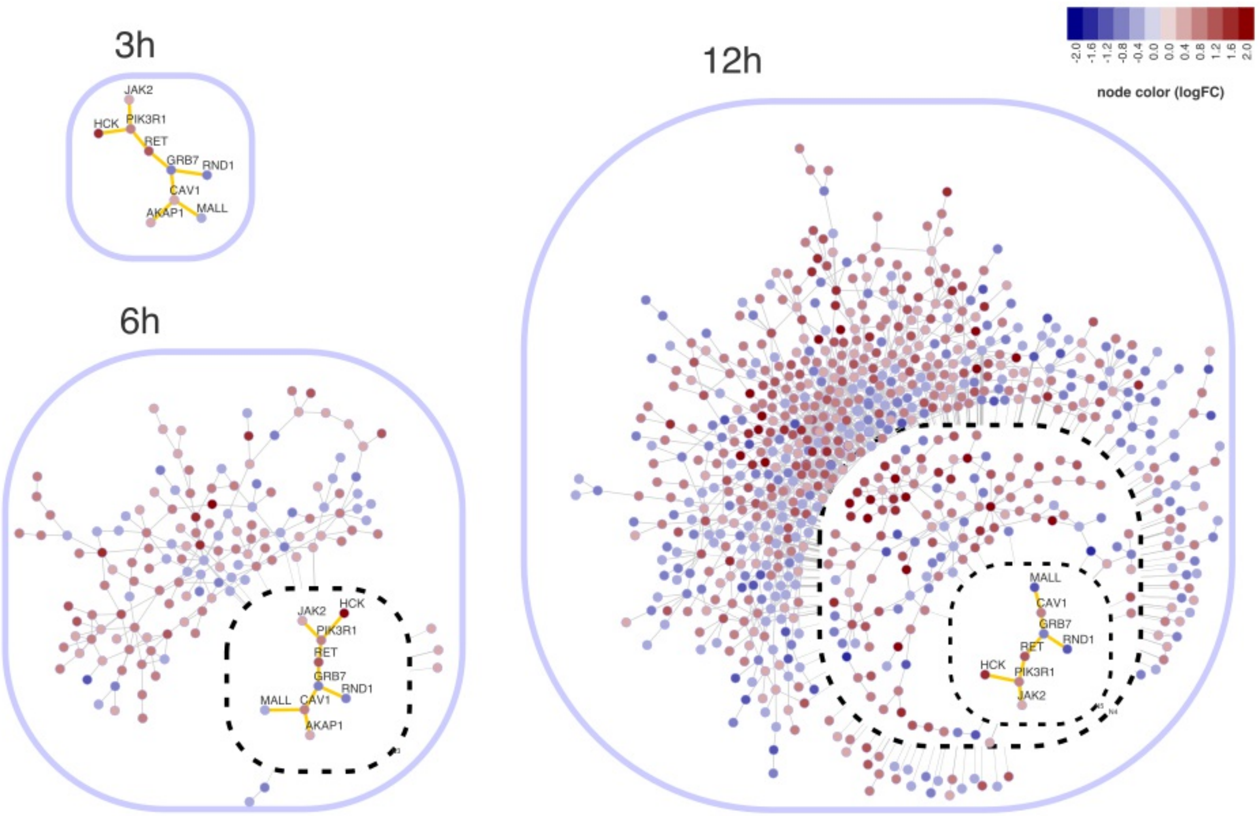


Figure 7. Nested subnetworks. This graph shows genes differentially expressed in estrogen-treated MCF-7 cells at 3, 6 and 12 h (relative to 0 h). The insets correspond to the overlap between consecutive time points (adapted from Castro et al. (2012))

3.2 Nested networks and clustering

Get the indiated data frame and igraph object:

```
data(ER.deg)
dt <- ER.deg$dat
sg <- ER.deg$ceg
```

Map the data frame to the graph:

```
sg <- att.mapv(sg, dat=dt, refcol=1)
```

Set attributes to the graph (i.e. gene symbols and two available numeric data):

```
sg <- att.setv(sg, from="Symbol", to="nodeAlias")
sg <- att.setv(sg, from="logFC.t3", to="nodeColor", breaks=seq(-1,1,0.2), pal=2)
sg <- att.setv(sg, from="ERbdist", to="nodeSize", nquant=10, isrev=TRUE, xlim=c(5,40,1))
```

Add graph to the app (**Fig.8**):

```
addGraph(rdp,sg)
```

Compute a hierarchical clustering using standard R functions:

```
hc <- hclust(dist(get.adjacency(sg, attr="weight")))
```

Map the `hclust` object onto the network and return corresponding ids (`pvclust` objects are also compatible):

```
nestID <- nesthc(rdp,hc, cutlevel=3, nmemb=5, cex=0.3, labels=V(sg)$nodeAlias)
```

...at this point nested objects from the network should appear mapped onto a dendrogram. Different levels of the nested structure can be set by the `nesthc` method. Additionally, clustering stability can be assessed by the **pvclust** package, which is already compatible with the interface.

Assign edges to containers:

```
mergeOutEdges(rdp,nlev=2)
```

Relax the network:

```
relax(rdp)
```

Add color and size legends:

```
scl <- sg$legNodeColor$scale
leg <- sg$legNodeColor$legend
addLegend.color(rdp, colvec=scl, labvec=leg, title="diff. gene expression (logFC)")


scl <- sg$legNodeSize$scale
leg <- sg$legNodeSize$legend
addLegend.size(rdp, sizevec=scl, labvec=leg, title="bd site distance (kb)")
```

Reset graph:

```
resetd(rdp)
```

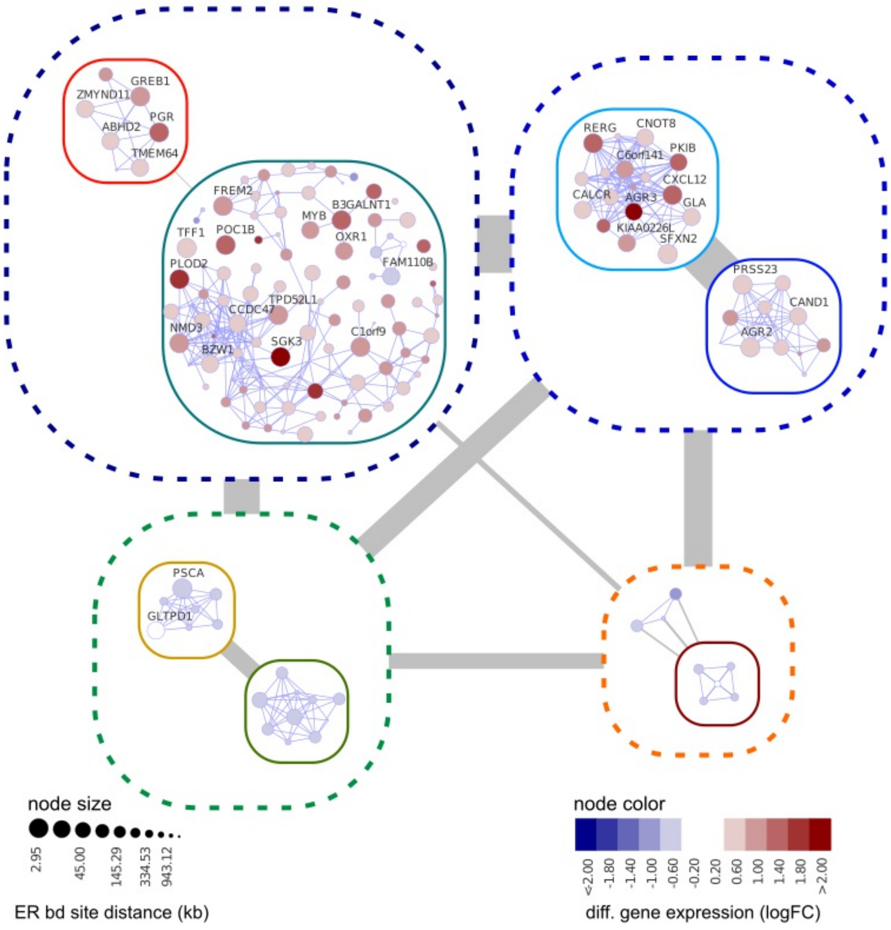


Figure 8. Hierarchical networks. This graph is obtained by superimposing a dendrogram onto the corresponding co-expression gene network (adapted from Castro et al. (2012)).

4 Installation

4.1 R package

The RedeR package is freely available from the Bioconductor at https://bioconductor.org/packages/RedeR/ (https://bioconductor.org/packages/RedeR/).

4.2 Java application

The RedeR jar file is already included in the R package and, as usual, to run Java applications your system should have a copy of the JRE (Java Runtime Environment, version >= 6).

5 Session information

```
## R version 3.4.2 (2017-09-28)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 16.04.3 LTS
##
## Matrix products: default
## BLAS: /home/biocbuild/bbs-3.6-bioc/R/lib/libRblas.so
## LAPACK: /home/biocbuild/bbs-3.6-bioc/R/lib/libRlapack.so
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=C
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] igraph_1.1.2    RedeR_1.26.0    BiocStyle_2.6.0
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.13    bookdown_0.5    lattice_0.20-35 digest_0.6.12
##  [5] rprojroot_1.2   grid_3.4.2      backports_1.1.1 magrittr_1.5
##  [9] evaluate_0.10.1 stringi_1.1.5   Matrix_1.2-11   rmarkdown_1.6
## [13] tools_3.4.2     stringr_1.2.0   yaml_2.1.14     compiler_3.4.2
## [17] pkgconfig_2.0.1 htmltools_0.3.6 knitr_1.17
```

References

Castro, Mauro AA, Xin Wang, Michael NC Fletcher, Kerstin B Meyer, and Florian Markowetz. 2012. “RedeR: R/Bioconductor Package for Representing Modular Structures, Nested Networks and Multiple Levels of Hierarchical Associations.” *Genome Biology* 13 (4): R29. doi:10.1186/gb-2012-13-4-r29 (https://doi.org/10.1186/gb-2012-13-4-r29).

Csardi, Gabor, and Tamas Nepusz. 2006. “The Igraph Software Package for Complex Network Research.” *InterJournal* Complex Systems: 1695. http://igraph.sf.net (http://igraph.sf.net).

“The Apache Software Foundation”. 2010. “Apache Xmlrpc Webserver.” http://ws.apache.org/xmlrpc/ (http://ws.apache.org/xmlrpc/).