RamiGO: an R interface for AmiGO

Markus S. Schröder[1,2], Daniel Gusenleitner[1], Aedín C. Culhane[1], Benjamin Haibe-Kains[1], and John Quackenbush[1]

[1] *Biostatistics and Computational Biology, Dana-Farber Cancer Institute, Harvard School of Public Health, Boston, USA*
[2] *Computational Genomics, Center for Biotechnology, Bielefeld University, Germany*

May 2, 2014

Contents

1 Introduction **1**

2 Getting started **2**

3 A usefull extension to GSEA **4**

4 View and edit GO trees in Cytoscape **5**

5 Misc **5**

6 Session Info **6**

1 Introduction

Please cite RamiGo as follows: Markus S. Schröder, Daniel Gusenleitner, John Quackenbush, Aedín C. Culhane and Benjamin Haibe-Kains (2013): RamiGO: an R/Bioconductor package providing an AmiGO Visualize interface. *Bioinformatics* 29(5):666-668.

A common task in recent gene set or gene signature analyses is testing for up- and down-regulation of these gene sets or gene signatures in Gene Ontology (GO) terms. Or having a gene or set of genes of interest and looking at the GO terms that include that gene or gene set. For a closer look at the distribution of the GO terms in the different tree structures of the three GO categories one has to either rebuild the GO tree himself with the help of published R packages, or copy and paste the GO terms of interest into existing web services to display the GO tree. One of these web services is AmiGO visualize:

AmiGO visualize: `http://amigo.geneontology.org/cgi-bin/amigo/amigo?mode=visualize`

The *RamiGO* package is providing functions to interact with the AmiGO visualize web server and retrieves GO (Gene Ontology) trees in various formats. The most common requests

would be as png or svg, but a file representation of the tree in the GraphViz DOT format is also possible. *RamiGO* also provides a parser for the GraphViz DOT format that returns a graph object and meta data in R.

2 Getting started

At first we load the *RamiGO* package into the current workspace:

```
> library(RamiGO)
```

The *RamiGO* package currently provides two functions that enable the user to retrieve directed acyclic trees from AmiGO and parse the GraphViz DOT format. An example on how to use the functions is given below.

To retrieve a tree from AmiGO, the user has to provide a vector of GO ID's. For example GO:0051130, GO:0019912, GO:0005783, GO:0043229 and GO:0050789. These GO ID's represent entries from the three GO categories: Biological Process, Molecular Function and Cellular Component. The given GO ID's can be highlighted with different colors within the tree, therefor the user has to provide a vector of colors for each GO ID. A request could look like this:

```
> goIDs <- c("GO:0051130","GO:0019912","GO:0005783","GO:0043229","GO:0050789")
> color <- c("lightblue","red","yellow","green","pink")
> pngRes <- getAmigoTree(goIDs=goIDs, color=color, filename="example", picType="png", savel
```

Figure 1: Example PNG returned from AmiGO.

The GO tree representing the given GO ID's is dowloaded to the file "example.png" (see Figure 1); the file extension is created automatically according to picType. The request for a svg file is similar:

```
> svgRes <- getAmigoTree(goIDs=goIDs, color=color, filename="example", picType="svg", savel
```

`svgRes` is a vector with the svg picture in xml format. In order to further analyze the tree, *RamiGO* provides the possibility to retrieve the tree in the GraphViz DOT format. The function `readAmigoDot` parses these DOT format files and returns a `AmigoDot` S4 object. This S4 object includes an igraph object (`agraph()`), an adjacency matrix representing the graph (`adjMAtrix()`), a data.frame with the annotation for each node (`annot()`), the relations (edges) between the nodes (`relations()`) and a data.frame with the leaves of the tree and their annotation (`leaves()`). An example could look like this:

```
> dotRes <- getAmigoTree(goIDs=goIDs, color=color, filename="example", picType="dot", savel
> tt <- readAmigoDot(object=dotRes)
> ## reading the file would also work!
> ## tt <- readAmigoDot(filename="example.dot")
> show(tt)
```

```
class: AmigoDot
Class 'igraph.es'  atomic [1:209] 1 2 3 4 5 6 7 8 9 10 ...
  ..- attr(*, "env")=<environment: 0x7ffbab49baa0>
nodes:
Class 'igraph.vs'  atomic [1:84] 1 2 3 4 5 6 7 8 9 10 ...
  ..- attr(*, "env")=<environment: 0x7ffbaf9b8158>
edges:
'data.frame':        5 obs. of  6 variables:
 $ node       : chr  "node4" "node14" "node40" "node70" ...
 $ GO_ID      : chr  "GO:0019912" "GO:0051130" "GO:0043229" "GO:0005783" ...
 $ description: chr  "cyclin-dependent protein kinase activating kinase activity" "positive
 $ color      : chr  "#000000" "#000000" "#000000" "#000000" ...
 $ fillcolor  : chr  "red" "lightblue" "green" "yellow" ...
 $ fontcolor  : chr  "#000000" "#000000" "#000000" "#000000" ...
leaves:
'data.frame':        84 obs. of  6 variables:
 $ node       : chr  "node1" "node2" "node3" "node4" ...
 $ GO_ID      : chr  "GO:0008150" "GO:0071840" "GO:0004672" "GO:0019912" ...
 $ description: chr  "biological_process" "cellular component organization or biogenesis"
 $ color      : chr  "#000000" "#000000" "#000000" "#000000" ...
 $ fillcolor  : chr  "#ffffff" "#ffffff" "#ffffff" "red" ...
 $ fontcolor  : chr  "#000000" "#000000" "#000000" "#000000" ...
annot:
'data.frame':        209 obs. of  6 variables:
 $ parent   : chr  "node1" "node1" "node1" "node1" ...
 $ child    : chr  "node2" "node7" "node13" "node35" ...
 $ arrowhead: chr  "none" "none" "none" "none" ...
 $ arrowtail: chr  "normal" "normal" "normal" "normal" ...
 $ color    : chr  "grey" "grey" "grey" "grey" ...
 $ style    : chr  "bold" "bold" "bold" "bold" ...
relations:
```

The leaves of the tree are returned in `leaves(tt)`:

```
> leavesTT <- leaves(tt)
> leavesTT[,c("node","GO_ID","description")]

     node      GO_ID
4   node4 GO:0019912
14 node14 GO:0051130
40 node40 GO:0043229
70 node70 GO:0005783
78 node78 GO:0050789
                                                  description
4  cyclin-dependent protein kinase activating kinase activity
14     positive regulation of cellular component organization
```

```
40                                  intracellular organelle
70                                    endoplasmic reticulum
78                           regulation of biological process
```

In order to export the tree to an GML file that is readable by Cytoscape, you have to call the `adjM2gml` with some of the results from the `readAmigoDot` function. The following example creates a GML file by internally calling the `exportCytoGML`:

```
> gg <- adjM2gml(adjMatrix(tt),relations(tt)$color,annot(tt)$fillcolor,annot(tt)$GO_ID,ann
```

The result is a GML file named example.gml that can be imported into Cytoscape as a network file.

3 A usefull extension to GSEA

The *RamiGO* package provides an extremely helpful extension to the GSEA software, in java as well as in R, if run with genesets from GO (C5 in MSigDB). *RamiGO* provides a mapping from GO terms returned from GSEA to official GO ID's. The mapping is stored in the data object `c5.go.mapping`.

```
> data(c5.go.mapping)
> head(c5.go.mapping)
```

```
                   description       goid
1                   NUCLEOPLASM GO:0005654
2 EXTRINSIC_TO_PLASMA_MEMBRANE GO:0019897
3               ORGANELLE_PART GO:0044422
4         CELL_PROJECTION_PART GO:0044463
5 CYTOPLASMIC_VESICLE_MEMBRANE GO:0030659
6               GOLGI_MEMBRANE GO:0000139
```

One of the ways to avoid running GSEA in R is to call the java application of GSEA from R with the `system()` function. An example for a preranked GSEA would be:

```
> ## paths to gsea jar and gmt file
> exe.path <- exe.path.string
> gmt.path <- gmt.path.string
> gsea.collapse <- "false"
> ## number of permutations
> nperm <- 10000
> gsea.seed <- 54321
> gsea.out <- "out-folder"
> ## build GSEA command
> gsea.report <- "report-file"
> rnk.path <- "rank-file"
> gsea.cmd <- sprintf("java -Xmx4g -cp %s xtools.gsea.GseaPreranked -gmx %s -collapse %s -
> ## execute command on the system
> system(gsea.cmd)
```

The results are stored in a folder with the name specified in `gsea.out`. The subfolder `gsea.report` has the detailed results in comma separated files and html pages. In the `gsea.cmd` string above we specified a few parameters which can be changed according to the type of analysis.

- plot_top_x: the number of results that should have an individual result page linked to the main index.html.
- set_max and set_min: limits the analysis to genesets that have more than 15 and less than 500 genes.

Once the GSEA analysis is finished, the important result files are xls files in the `gsea.report` folder. Named gsea_report_for_na_pos_<some number>.xls and gsea_report_for_na_neg_<some number>.xls. We can read them into R with the following command:

```
> resn <- "xxx" ## number generated by GSEA that you can get with grep(), strsplit() and d
> tt <- rbind(read.table(sprintf("%s/%s/gsea_report_for_na_pos_%s.xls", gsea.out, gsea.rep
```

With all results from the GSEA analysis stored in `tt`, you can extract information from the results and call the `getAmigoTree` mentioned in the example section.

4 View and edit GO trees in Cytoscape

The `adjM2gml` function in *RamiGO* creates a Cytoscape specific GML file (see example section above) that can be imported into Cytoscape and further edited (for example for publication purposes). The GO tree from the example above, parsed with the `readAmigoDot` function, exported with the `adjM2gml` and imported into Cytoscape as a network, looks like Figure 2.

5 Misc

`strapply` enables perl-like regular expression in R, as do `grep, sub or gsub`. In particular, it enables the use of the perl variables $1, $2, ... for extracting information from within a regular expression. The code below shows an example of the use of `strapply`. The string within brackets (...) is returned in a list by `strapply`.

```
> strapply(c("node25 -> node30"), "node([\\d]+) -> node([\\d]+)", c, backref = -2)

[[1]]
[1] "25" "30"
```

The *RCurl* package is useful for communicating with a web server and sending GET or POST requests. *RamiGO* uses the `postForm()` function to communicate with the AmiGO web server. The *png* package is used to convert the web server response for a png request into an actual png file. The *igraph* package is used to build a graph object representing the tree that was parsed from an DOT format file.

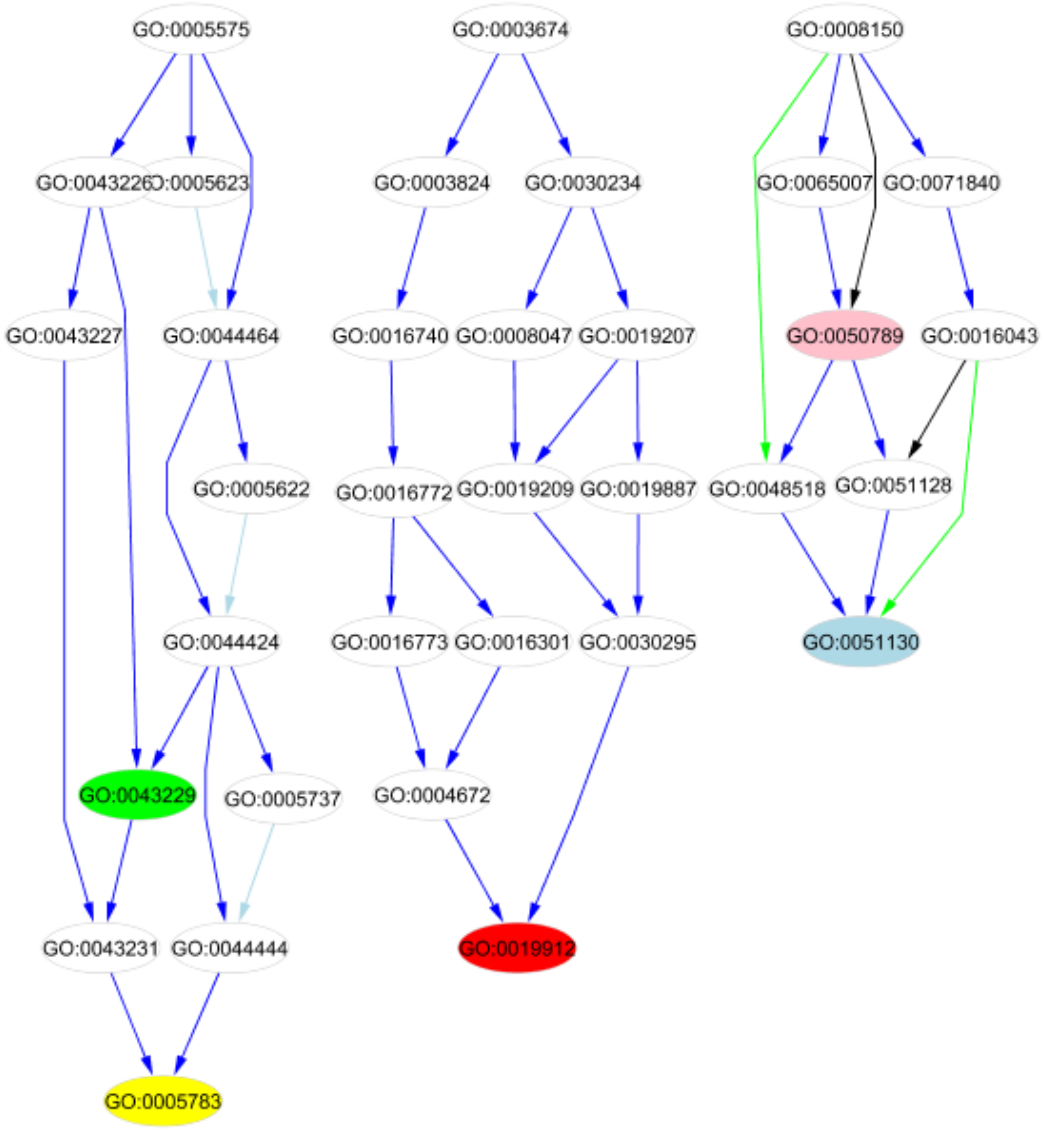

Figure 2: Example GML imported in Cytoscape.

6 Session Info

- R version 3.1.0 (2014-04-10), `x86_64-apple-darwin13.1.0`
- Locale: `C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8`
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: RamiGO 1.10.0, gsubfn 0.6-5, proto 0.3-10
- Loaded via a namespace (and not attached): BiocGenerics 0.10.0, RCurl 1.95-4.1,

RCytoscape 1.14.0, XML 3.98-1.1, XMLRPC 0.3-0, graph 1.42.0, igraph 0.7.1, parallel 3.1.0, png 0.1-7, stats4 3.1.0, tcltk 3.1.0, tools 3.1.0