

RCy3

Tanja Muetze, Georgi Kolishovski, Paul Shannon

October 30, 2017

Cytoscape is a well-known bioinformatics tool for displaying and exploring biological networks. **R** is a powerful programming language and environment for statistical and exploratory data analysis. *RCy3* uses CyREST to communicate between **R** and Cytoscape, allowing Bioconductor graphs to be viewed, explored and manipulated with the Cytoscape point-and-click visual interface. Thus, via RCy3, these two quite different, quite useful bioinformatics software environments are connected, mutually enhancing each other, providing new possibilities for exploring biological data.

1 Prerequisites

In addition to this package (RCy3), you will need:

- Java SE 8. Cytoscape 3.3 and CyREST currently do not support Java SE 6 or 7 anymore. Java SE 7 works only in combination with Cytoscape 3.2. Java can be downloaded from Oracle here: <http://www.oracle.com/technetwork/java/javase/downloads/index.html>.
- Cytoscape version 3.3+, which can be downloaded from <http://www.cytoscape.org/download.php>.
- CyREST, version 3.0.0 or later, a Cytoscape plugin which provides the Cytoscape end of the communication layer, can be installed via the App Manager or downloaded from the Cytoscape plugins website: <http://apps.cytoscape.org/apps/cyrest>.

2 Getting Started

First install Java, then Cytoscape and follow the instructions on the screen. To install CyREST, you can either open Cytoscape, click on “Apps” in the menubar and select “App Manager”. In the App Manager on the “Install Apps” tab, type “CyREST” in the search bar and click on CyREST when it appears and then click on “Install”. Alternatively

you can download the CyREST plugin as .jar file from Cytoscape’s app store here: <http://apps.cytoscape.org/apps/cyrest>. Then, open Cytoscape, click on “Apps” in the menubar and select “App Manager”. In the App Manager on the “Install Apps” tab, click on “Install from File...”, choose the .jar file that you just downloaded, then click “Open” and then “Install”. You are all set!

3 A minimal example

Here we create a 3-node graph in R, send it to Cytoscape for display and layout. For the sake of simplicity, no node attributes, no edges, and no vizmapping is included; those topics are covered in subsequent examples.

```
> library(RCy3)
> g <- new ('graphNEL', edgemode='directed')
> g <- graph::addNode ('A', g)
> g <- graph::addNode ('B', g)
> g <- graph::addNode ('C', g)
> cw <- CytoscapeWindow ('vignette', graph=g, overwrite=TRUE)
> displayGraph (cw)
>
```

You should now see a single red dot in the middle of a small Cytoscape window titled ‘simple’, contained along with possibly other windows within the Cytoscape Desktop. This graph needs layout. You can accomplish this in two ways: by function call (see below), or interactively from the Cytoscape ‘Layout’ menu, where a reasonable choice is ‘JGraph Layouts’ -> ‘GEM Layout’, or ‘yFiles Organic’.

After layout, you will see the structure of this graph: 3 unconnected nodes, unlabeled, small and colored red, and thus not very informative. Fortunately, Cytoscape has some built-in rendering rules in which (and unless instructed otherwise) nodes are rendered round in shape, pale red in color, and labeled with the names supplied when they were added to the graph; edges (if any) are simple blue lines. To get this default rendering, simply type.

```
> layoutNetwork (cw, layout.name='grid')
>
```

4 Add node attributes

We often know quite a lot about the nodes and edges in our graphs. By conveying this information visually, the graph will be easier to explore. For instance, we may know that protein A phosphorylates protein B, that A is a kinase and B a transcription factor, and that their mRNA expression (compared to a control) is a log fold change, base 2, of

1.8 and 3.2 respectively. One of the core features of Cytoscape, the ‘vizmapper’, allows you to specify how data values (e.g., ‘kinase’, ‘transcription factor’; expression ratios) should control the visual attributes of the graph (node shape, node color or size). Here is a simple example. (Note: edge attributes behave just like node attributes. They can be assigned, communicated along with their graph to R, and used to control visual attributes of the graph. In the current exercise, however, to keep things simple, only node attributes are discussed. Edges and edge attributes are discussed below.)

We continue with the small 3-node graph created in the previous code example, adding two kinds data attributes (‘moleculeType’ and ‘lfc’). Note, however, that adding the attributes does not in itself cause the appearance of the graph to change. Such a change requires that you specify and apply the visual mapping rules, which will be explained in the *next* section.

You can, however, examine these attributes in Cytoscape, using Cytoscape’s the ‘Data Panel’ to display attribute values of all nodes currently selected in the Cytoscape window:

- Select a node or nodes with your mouse (or programmatically, from R, using the ‘selectNodes’ function)
- Open up Cytoscape’s *Data Panel* (from the View menu)
- The attributes are displayed in a spreadsheet-like format in the Data Panel

You will encounter one obscurity in the code chunk just below: the explicit initialization of node (and edge) attributes for a graph. This is similar to, but goes a bit further than, calls to *nodeDataDefaults* required for Bioconductor graph objects. In fact, such calls are executed within the RCy3 *initNodeAttribute* and *initEdgeAttribute* methods. An additional bit of information is required here: the ‘class’ or data type of the attribute you are initializing. It can be ‘char’, ‘integer’ or ‘numeric’.

For instance, the node attribute ‘moleculeType’ has values which are character strings. The node attribute ‘lfc’ (log fold change) is declared to be numeric. This extra fuss is unavoidable: if you do not stipulate the class of each node and edge attribute, as is demonstrated below, then RCy3 will report an error. We require this because it is only by way of such explicit assignment that RCy3 can resolve the difference between integer and floating point values – necessary in the statically typed Java language (in which Cytoscape is written) though not in R.

Here is the code for adding attributes to the nodes of the 3-node graph whose creation is demonstrated above. Note that one usually creates the structure of a graph – specify its nodes and edges – initializes node and edge attributes, specifies values for those attributes, and only then displays the graph. For tutorial purposes here, the already-displayed graph is modified and redisplayed.

```
> g <- cw@graph # created above, in the section 'A minimal example'  
> g <- initNodeAttribute (graph=g, attribute.name='moleculeType',
```

```

+                                     attribute.type='char',
+                                     default.value='undefined')
> g <- initNodeAttribute (graph=g, 'lfc', 'numeric', 0.0)
> nodeData (g, 'A', 'moleculeType') <- 'kinase'
> nodeData (g, 'B', 'moleculeType') <- 'TF'
> nodeData (g, 'C', 'moleculeType') <- 'cytokine'
> nodeData (g, 'A', 'lfc') <- -1.2
> nodeData (g, 'B', 'lfc') <- 1.8
> nodeData (g, 'C', 'lfc') <- 3.2
> cw <- setGraph (cw, g)
> displayGraph (cw) # cw's graph is sent to Cytoscape
>

```

You can now explore these simple node attribute values in Cytoscape, selecting nodes, and navigating around inside the Cytoscape ‘Data Panel’ – which is designed for exactly that purpose.

5 Modifying the display: default values and mapping rules

By default, Cytoscape displays nodes as pale red circles circles, and edges as blue undecorated lines. RCy3 provides an easy way to change these defaults. More interestingly, RCy3 provides easy programmatic access to the *vizmapper*, by means of which the size, shape and color of nodes and edges is determined by the data attributes you have attached to those nodes and edges.

First, let’s change the the defaults. Note that ‘redraw’ must be called on the CytoscapeWindow to force an update of the display. Cytoscape is a little inconsistent about this: setting defaults, like you see below, requires the redraw, but setting rules (about which more below) does not. We hope to get more consistency of Cytoscape soon. In the meantime, if you are rendering a larg graph, you may want to call this only when you are sure you need to.

```

> setDefaultNodeShape (cw, 'OCTAGON')
> setDefaultNodeColor (cw, '#AAFF88')
> setDefaultNodeSize (cw, 80)
> setDefaultNodeFontSize (cw, 40)

```

Now we will add some visual mapping – (*vizmap*) – rules. These rules connect data attributes to visual attributes, thereby giving the eye with additional information about the network. In our first mapping, we use map from the data attribute ‘moleculeType’ to node shapes to. To begin, ask Cytoscape for the node shape names it recognizes. Then query the graph for a list of the data attributes, establishing that ‘moleculeType’

is indeed a current data attribute of all nodes. Then find out what distinct values are defined for `moleculeType` across the nodes of the graph. Finally, define and set the rule.

```
> getNodeShapes (cw) # diamond, ellipse, trapezoid, triangle, etc.
> print (noa.names (getGraph (cw))) # what data attributes are defined?
> print (noa (getGraph (cw), 'moleculeType'))
> attribute.values <- c ('kinase', 'TF', 'cytokine')
> node.shapes <- c ('DIAMOND', 'TRIANGLE', 'RECTANGLE')
> setNodeShapeRule (cw, node.attribute.name='moleculeType',
+ attribute.values, node.shapes)
```

The node shape rule, just above, is an example of a ‘lookup’ rule, which is sometimes called a ‘discrete’ rule. The network has three nodes, each of them a gene, and each of them has a ‘moleculeType’ attribute. In this case, the values are ‘kinase’, ‘TF’ (for transcription factor) and ‘cytokine’. These are the discrete values taken on by the `moleculeType` node attribute. For each, we specify the shape we wish to use in rendering that type of molecule. Thus, there is a discrete set of values, and we map from those values to shapes by a simple ‘lookup’ process.

A second class of rules – in contrast to lookup rules – uses interpolation. In the classic example, every node (every gene) has a mRNA expression value, expressed as a log fold change (‘lfc’) ratio, experiment vs. control. Nodes with negative lfc are rendered in shades of green; nodes with positive lfc are rendered in shades of red. Nodes with `lfc == 0` are rendered in white. All intermediate values are rendered in appropriately interpolated shades of either green or red.

`setNodeColorRule` and `setNodeSizeRule` are commonly interpolation rules: use `mode='interpolate'`. But they may also be called as lookup rules: use parameter `mode='lookup'` to accomplish this, and provide as many data points and sizes (or colors) needed to cover all possible values of the attribute you are mapping.

```
> setNodeColorRule (cw, 'lfc', c (-3.0, 0.0, 3.0),
+ c ('#00AA00', '#00FF00', '#FFFFFF', '#FF0000', '#AA0000'),
+ mode='interpolate')
```

Note that there *five* colors, but only three control.points. The two additional colors tell the interpolated mapper which colors to use if the stated data attribute (lfc) has a value less than the smallest control point (paint it a darkish green, `#00AA00`) or larger than the largest control point (paint it a darkish red, `#AA0000`). These extreme (or out-of-bounds) colors may be omitted:

```
> setNodeColorRule (cw, 'lfc', c (-3.0, 0.0, 3.0),
+ c ('#00FF00', '#FFFFFF', '#FF0000'),
+ mode='interpolate')
```

in which case RCy3 will reuse the first and last values (green and red) for out-of-bounds values, and issue a warning to the console.

Now, add a node size rule, using 'lfc' again as the controlling node attribute.

```
> control.points = c (-1.2, 2.0, 4.0)
> node.sizes      = c (10, 20, 50, 200, 205)
> setNodeSizeRule (cw, 'lfc', control.points, node.sizes,
+                 mode='interpolate')
>
```

6 Add some edges, edge attributes, and some rules for their rendering.

```
> g <- cw@graph
> g <- initEdgeAttribute (graph=g, attribute.name='edgeType',
+                         attribute.type='char',
+                         default.value='unspecified')
> g <- graph::addEdge ('A', 'B', g)
> g <- graph::addEdge ('B', 'C', g)
> g <- graph::addEdge ('C', 'A', g)
> edgeData (g, 'A', 'B', 'edgeType') <- 'phosphorylates'
> edgeData (g, 'B', 'C', 'edgeType') <- 'promotes'
> edgeData (g, 'C', 'A', 'edgeType') <- 'indirectly activates'
> cw@graph <- g
> displayGraph (cw)
> line.styles = c ('DOT', 'SOLID', 'SINEWAVE')
> edgeType.values = c ('phosphorylates', 'promotes',
+                     'indirectly activates')
> setEdgeLineStyleRule (cw, 'edgeType', edgeType.values,
+                       line.styles)
> arrow.styles = c ('Arrow', 'Delta', 'Circle')
> setEdgeTargetArrowRule (cw, 'edgeType', edgeType.values,
+                          arrow.styles)
```

7 Hide, Show, and Float Cytoscape Panels

If you want to have more of the Cytoscape Desktop devoted to displaying your graph, you can hide the panels which normally occupy the left and bottom proting of that Desktop. There are related panels for 'docking' and 'floating' those panels. To save on typing, your arguments to these functions (see below) can be very terse, and are case-independent.

```

> hidePanel (cw, 'Data Panel')
> floatPanel (cw, 'D')
> dockPanel (cw, 'd')
> hidePanel (cw, 'Control Panel')
> floatPanel (cw, 'control')
> dockPanel (cw, 'c')

```

8 Selecting Nodes

Let us now try some simple back-and-forth between Cytoscape and R. In Cytoscape, click the ‘B’ node. In R:

```

> getSelectedNodes (cw)

```

Now we wish to extend the selected nodes to include the first neighbors of the already-selected node ‘B’. This is a common operation: for instance, after selecting one or more nodes based on experimental data or annotation, you may want to explore these in the context of interaction partners (in a protein-protein network) or in relation to upstream and downstream partners in a signaling or metabolic network. Type:

```

> selectFirstNeighborsOfSelectedNodes (cw)

```

You will see that all three nodes are now selected. Get their identifiers back to R:

```

> nodes <- getSelectedNodes (cw)

```

9 Positioning nodes (and possibilities for data-driven graph layout)

Despite its simplicity, RCy3’s *setNodePosition* method provides the means to create custom layout algorithms and animations. Here is a rather nonsensical example, in which the ‘A’ node orbits around a fixed center, with the ‘B’ and ‘C’ nodes left unchanged.

```

> cwe <- CytoscapeWindow ('vignette.setNodePosition',
+                          graph=RCy3::makeSimpleGraph (), overwrite=TRUE)
> displayGraph (cwe)
> layoutNetwork (cwe, 'grid')
> center.x <- 200
> center.y <- 200
> radius <- 200
> # sweep through full revolution 3 times, 5 degrees at a time
> angles <- seq (0, 360, 90)

```

```

>   for (angle in angles) {
+     angle.in.radians <- angle * pi / 180
+     x <- center.x + (radius * cos (angle.in.radians))
+     y <- center.y + (radius * sin (angle.in.radians))
+     setNodePosition (cwm, 'A', x, y)
+   }
>   # RCy3 will not create windows with duplicate names, so clear the decks for a s

```

10 Movies: explore time course data by animating node color and size

Time course microarray expression experiments are commonplace in molecular biology research. Once reduced, the expression data is often summarized as a lfc (log fold change, experiment vs control) and a p-value for each gene at each time point. Many Bioconductor packages can be used to find biologically interesting patterns in such data, but visualization – as one aspect of exploratory data analysis – should not be overlooked. Here is a toy problem, a small example illustrating how this can be accomplished with RCy3.

We begin with the small sample graph just like that created in the ‘setNodePosition’ example above. We add some made-up statistical significance data into a new node attribute, ‘pval’.

```

>   g <- RCy3::makeSimpleGraph ()
>   g <- initNodeAttribute (g, 'pval', 'numeric', 1.0)
>   cwm <- CytoscapeWindow ('movie', graph=g, overwrite=TRUE)
>   displayGraph (cwm)
>   layoutNetwork (cwm, 'grid')

```

Now define the node size and node color rules we will use in the animation:

```

>   lfc.control.points <- c (-3.0, 0.0, 3.0)
>   lfc.colors <- c ('#00AA00', '#00FF00', '#FFFFFF', '#FF0000', '#AA0000')
>   setNodeColorRule (cwm, 'lfc', lfc.control.points, lfc.colors,
+                     mode='interpolate')
>   pval.control.points <- c (0.1, 0.05, 0.01, 0.0001)
>   pval.sizes <- c (30, 50, 70, 100)
>   setNodeSizeRule (cwm, 'pval', pval.control.points, pval.sizes,
+                    mode='interpolate')

```

Once the rules are in effect, node color and size will always be a function of each nodes’ lfc and pval numeric attributes. Animation is accomplished easily: all you have to do is to update these two attributes for each node at each time point, and asking Cytoscape

to render the nodes following the rules it already has been instructed to follow. Note that a message is written to the Cytoscape status message bar – found at the lower left corner of the Cytoscape desktop – to help keep track of what the current timepoint is.

```

> pval.timepoint.1 <- c (0.01, 0.3, 0.05)
> pval.timepoint.2 <- c (0.05, 0.01, 0.01)
> pval.timepoint.3 <- c (0.0001, 0.005, 0.1)
> lfc.timepoint.1 <- c (-1.0, 1.0, 0.0)
> lfc.timepoint.2 <- c (2.0, 3.0, -2.0)
> lfc.timepoint.3 <- c (2.5, 2.0, 0.0)
> for (i in 1:5) { # run this loop 5 times
+   setNodeAttributesDirect (cwm, 'lfc', 'numeric', c ('A', 'B', 'C'),
+                           lfc.timepoint.1)
+   setNodeAttributesDirect (cwm, 'pval', 'numeric', c ('A', 'B', 'C'),
+                           pval.timepoint.1)
+   system ('sleep 0.3')
+
+   setNodeAttributesDirect (cwm, 'lfc', 'numeric', c ('A', 'B', 'C'),
+                           lfc.timepoint.2)
+   setNodeAttributesDirect (cwm, 'pval', 'numeric', c ('A', 'B', 'C'),
+                           pval.timepoint.2)
+   system ('sleep 0.3')
+
+   setNodeAttributesDirect (cwm, 'lfc', 'numeric', c ('A', 'B', 'C'),
+                           lfc.timepoint.3)
+   setNodeAttributesDirect (cwm, 'pval', 'numeric', c ('A', 'B', 'C'),
+                           pval.timepoint.3)
+   system ('sleep 0.3')
+ }
>

```

Though this example is very simple and quite without biological meaning, we hope that it adequately illustrates the technique. We commonly use just this approach to explore large scale time course expression data experiments in the context of curated, biologically relevant pathways, such as those provided by KEGG, and selected by packages such as SPIA or Category.

Finally, delete all the windows from this vignette:

```

> cy <- CytoscapeConnection ()
> window.names <- c ('vignette', 'vignette.setNodePosition', 'movie')
> for (window.name in window.names){
+   if (window.name %in% as.character (getWindowList (cy))){
+     deleteWindow (cy, window.name)
+   }
+ }

```

```
+      }  
+    }
```

11 Online Documentation

https://github.com/tmuetze/Bioconductor_RCy3_the_new_RCytoscape/tree/master/man

12 Future Work

- All vizmap rules are currently added to the ‘default’ vizmap. It is not yet possible to delete rules from this map, either selectively or as a clean sweep. More flexible use of vizmaps is high on the to-do list.

13 Credits

- Paul Shannon’s generous advice and mentorship was very important for transforming this package from using XMLRPC and CytoscapeRPC to using CyREST.
- Keiichiro Ono and Barry Demchak kindly provided CyREST as well as help and support for new functionalities and changes.
- Mark Grimes kindly provided helpful user feedback.
- Julia Gustavsen generously developed various use cases/examples for using RCy3 with biological data during GSOC 2016. More detail can be found here: https://github.com/joolia/gsoc_Rcy3_vignettes/blob/master/blog_post_drafts/final_work_submission.md

14 References

- Shannon P, Markiel A, Ozier O, Baliga NS, Wang JT, Ramage D, Amin N, Schwikowski B, Ideker T. 2003. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome Res.* Nov;13(11):2498-504
- Huber W, Carey VJ, Long L, Falcon S, Gentleman R. 2007. Graphs in molecular biology. *BMC Bioinformatics.* 2007 Sep 27;8.
- Ono K, Muetze T, Kolishovski G, Shannon P, Demchak, B. CyREST: Turbocharging Cytoscape Access for External Tools via a RESTful API [version 1; referees: 2 approved]. *F1000Research* 2015, 4:478.