Overview of RCy3

by Alexander Pico, Tanja Muetze, Georgi Kolishovski, Paul Shannon

2018-03-17

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 any graphs (e.g., iGraph, graphNEL or dataframes) 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.

Prerequisites

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

- **Cytoscape 3.6.1** or greater, which can be downloaded from http://www.cytoscape.org/download.php. Simply follow the installation instructions on screen.

Getting started

First, launch Cytoscape and keep it running whenever using RCy3. Confirm that you have everything installed and running:

```
cytoscapePing ()
```

Simple example

Here we create a 4-node graph in R, send it to Cytoscape for display and layout. For the sake of simplicity, no node attributes and no visual styles are included; those topics are covered in subsequent steps.

```
g = new ('graphNEL', edgemode='directed')
g = graph::addNode ('A', g)
g = graph::addNode ('D', g)
g = graph::addNode ('C', g, edges = list('D'))
g = graph::addNode ('B', g, edges = list(c('A','D','C')))
createNetworkFromGraph (g)
```

You should now the structure of this 4-node graph with a basic, default style. Fortunately, Cytoscape has some built-in rendering rules in which (and unless instructed otherwise) nodes and edges are rendered and a default (user-preference) layout algorithm is applied.

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 log2 fold change of 1.8 and 3.2 respectively. One of the core features of Cytoscape is visual styles, which allow you to specify how data values (e.g., `kinase',`transcription factor'; expression ratios) should be conveyed in the visual properties of the graph (e.g., node shape, node color or size).

We continue with the simple 4-node graph, adding two kinds data values (`moleculeType' and`log2fc'). The easiest way to do this is via data.frames. However, you can also include attributes together with the original graph models as Bioconductor graphs, iGraphs or data.frames and then use the provided *create* functions to create and load in a singel step (see *createNetworkFromGraph*, *createNetworkFromIgraph* and *createNetworkFromDataFrames*)

```
df <- data.frame (moleculeType=c('kinase','TF','cytokine','cytokine'),
                 log2fc=c(1.8,3.0,-1.2,-2.5),
                 row.names = c('A','B','C','D'), # row.names = node names
                 stringsAsFactors = FALSE)       # important when loading strings!
loadTableData (df)
```

Note, 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 visual style mappings, which will be explained in the *next* section. You can, however, examine these attributes in Cytoscape, using Cytoscape's the **Data Panel** to display data values associated with selected nodes immediately below the Cytoscape window.

Modifying the display: defaults and mappings

RCy3 provides an easy way to not only change the default styles, but more interestingly, RCy3 also provides easy access to *mapping* your data to visual styles, e.g., allowing the size, shape and color of nodes and edges to be determined by the data you have associated with those nodes and edges.

First, let's change the the defaults.

```
setNodeShapeDefault ('OCTAGON')
setNodeColorDefault ('#AAFF88')
setNodeSizeDefault  (60)
setNodeFontSizeDefault (30)
```

Now we will add some visual mappings. Let's map `moleculeType' to node shapes. First, we can see which shapes are available in Cytoscape, then we can define the mapping with paired lists.

```
getNodeShapes ()   # diamond, ellipse, trapezoid, triangle, etc.
```

```
column <- 'moleculeType'
values <- c ('kinase',  'TF','cytokine')
shapes <- c ('DIAMOND', 'TRIANGLE', 'RECTANGLE')
setNodeShapeMapping (column, values, shapes)
```

The node shape mapping is an example of a *discrete* mapping, where a style is defined for each, discrete value. This is useful for categorical data (like type) where there is only a limited set of possible values. This is in contast to the other two other types of mappings: *continous* and *passthrough*. In the case of expression values, for example, we will want to use *contiuous* mapping (e.g., to node color), defining a small set of control points, rather than an explicit color for each possible data value. Cytoscape will simply interpolate between the control points to provide a gradient of colors. Let's try that one now

```
column <- 'log2fc'
control.points <- c (-3.0, 0.0, 3.0)
colors <-  c ('#5588DD', '#FFFFFF', '#DD8855')
setNodeColorMapping (column, control.points, colors)
```

Note that there are three colors and three control points. However, you can also specify *two additional* colors beyond the number of control points if you want to set extreme (or out-of-bounds) colors for values less than or greater than your control points.

```
control.points <- c (-2.0, 0.0, 2.0)
colors <-  c ('#2255CC', '#5588DD', '#FFFFFF', '#DD8855','#CC5522')
setNodeColorMapping (column, control.points, colors)
```

Now, add a node size rule, using *log2fc* again as controlling node values.

```
control.points = c (-3.0, 2.0, 3.0)
sizes     = c (20, 80, 90)
setNodeSizeMapping (column, control.points, sizes)
```

If you recall the third type of mapping, *passthrough* we can see it already working in our current network example. The node labels! By default, the *name* column is mapped to the node label property using *passthrough* logic: the value is passed directly to the style property.

Selecting nodes

Let us now try selecting nodes in Cytoscape from R. Select the C node by name:

```
selectNodes ('C','name')
```

```
getSelectedNodes ()
```

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:

```
selectFirstNeighbors ()
```

You will see that three nodes are now selected. Get their SUID identifers back to R as a list:

```
node.suids <- getSelectedNodes ()
```

Saving and Export

Session files save *everything*. As with most project software, we recommend saving often!

```
saveSession('tutorial_session') #.cys
```

Note: If you don't specify a complete path, the files will be save relative to your Cytoscape installation directory, e.g., /Applications/Cytoscape_v3.6.0/… or somewhere you don't have write permissions. So, it's probably better to specify one…

```
full.path=paste(getwd(),'tutorial_session',sep='/')
saveSession(filename=full.path) #.cys
```

Saving high resolution image files

You can export extremely high resolution images, including vector graphic formats.

```
full.path=paste(getwd(),'tutorial_image2',sep='/')
exportImage(filename=full.path, type = 'PDF') #.pdf
?exportImage
```

That covers the basics of network manipulation. Check out the other vignettes for additional amd more complex examples.

Development

The RCy3 project code and documentation is maintained at GitHub: https://github.com/cytoscape/RCy3. All bugs and feature requests are tracked as **Issues**, https://github.com/cytoscape/RCy3/issues.

Credits

- Paul Shannon's generous advice and mentorship was very important for transforming this

package from using XMLRPC and CytoscapeRPC to using CyREST.
- David Otasek, Keiichiro Ono and Barry Demchak kindly provided CyREST as well as help and support for new functionalities and changes.
- Mark Grimes and Ruth Isserlin kindly provided helpful user feedback.
- Julia Gustavsen generously developed various use cases/examples for using RCy3 with biological data during GSOC 2016, https://github.com/jooolia/gsoc_Rcy3_vignettes/blob/master/blog_post_drafts/final_work_submission.md.
- Tanja Muetze provided many years of development, design, maintenance and documentation for the RCy3 project.
- All contributors, new and old, are dynamically acknowledged in our **Contributor Graph**, https://github.com/cytoscape/RCy3/graphs/contributors

References

1. 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
2. Huber W, Carey VJ, Long L, Falcon S, Gentleman R. 2007. Graphs in molecular biology. BMC Bioinformatics. 2007 Sep 27;8.
3. 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.