RCyjs: using Cytoscape.js from an R session

Paul Shannon

May 3, 2018

Contents

1 Introduction

RCyjs is a *BrowserViz* subclass providing interactive access to Cytoscape.js running in your browser from an R session. Cytoscape.js http://js.cytoscape.org provides full-featured network visualization in an HTML5 browser window. This web-based implementation is related to, and can exchange data with the desktop version of Cytoscape: http://cytoscape.org. See the *BrowserViz* vignette for a description of the websocket and JSON techniques used here to connect your R session to your browser.

2 Simple Example

RCyjs provides a utility function which creates a small 3-node, 3-edge graphNEL (see *graph*) with some attributes assigned to the nodes and edges:
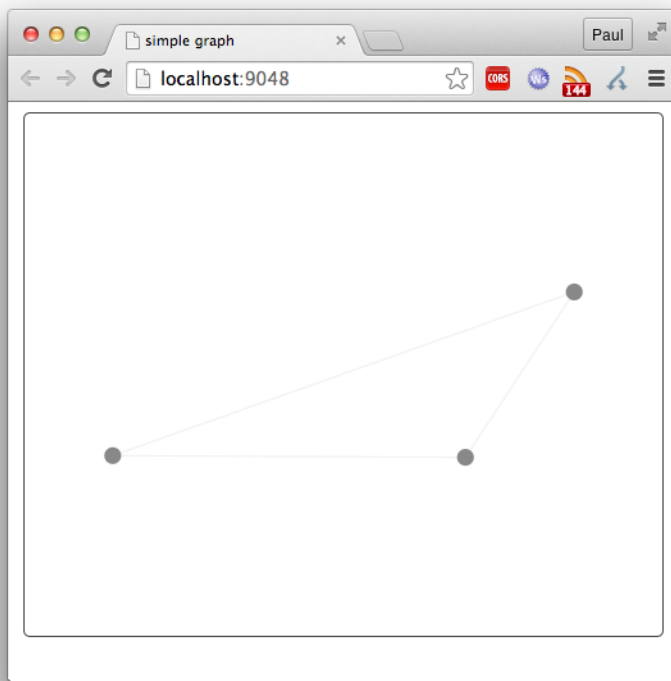
```
> library(RCyjs)
> g <- simpleDemoGraph()
> noaNames(g)
> edaNames(g)
> noa(g, "type")
> noa(g, "lfc")
> eda(g, "edgeType")
> eda(g, "score")
```

Send this graph to your browser and render it with all default settings (including faint edges connecting the nodes):

```
> g <- simpleDemoGraph()
> rcy <- RCyjs(portRange=9047:9067, quiet=TRUE, graph=g);
```

```
> title <- "simple graph"
> setBrowserWindowTitle(rcy, title)
```


In network visualization we commonly use edge and node attributes to control visual appearance. For instance, an edgeType of "phosphorylates" could be rendered in green, and "regulates" in black. Node size can reflect expression levels or population size. Node borders can indicate abnormal modifications. This process, and the rules that govern it, go by the name of "visual mapping".

Before discussing and illustrating the use of these rules, let us first see how to set default visual properties which apply to all parts of the graph. For instance: make all nodes round, with white interior, a thin black border, 50 pixels in diameter, connected by black edges (lines) which have no terminal decoration (that is, no arrows at the ends of the edges). Note that *redraw* must be called to *apply* the currently specified rules. First we define some useful colors:

```
> green <- "rgb(0,255,0)"
> white <- "rgb(255,255,255)"
> red <- "rgb(255,0,0)"
> blue <- "rgb(0,0,255)"
> black <- "rgb(0,0,0)"
> darkGreen <- "rgb(0,200,0)"
> darkerGreen <- "rgb(0,120,0)"
> darkRed <- "rgb(221,0,0)"
> darkerRed <- "rgb(170,0,0)"
```

```
> purple <- "rgb(221,221,0)"
> darkBlue <- "rgb(0,0,170)"
> darkerBlue <- "rgb(0,0,136)"
> lightGray="rgb(230,230,230)"
>
```

```
> setDefaultNodeShape(rcy, "ellipse")
> setNodeLabelRule(rcy, "label")  # we DO want a different label on each node.  this rule ensures that
> setDefaultNodeSize(rcy, 70)
> setDefaultNodeColor(rcy, "white")
> setDefaultNodeBorderColor(rcy, "black")
> setDefaultNodeBorderWidth(rcy, 1)
> setDefaultEdgeColor(rcy, blue)
> setNodeLabelAlignment(rcy, "center", "center");
> layout(rcy, "circle")
> fitContent(rcy)
> setZoom(rcy, 0.75 * getZoom(rcy))
> setDefaultEdgeSourceArrowShape(rcy, "tee")
> setDefaultEdgeTargetArrowShape(rcy, "triangle")
> setDefaultEdgeSourceArrowColor(rcy, blue)
> setDefaultEdgeTargetArrowColor(rcy, blue)
> redraw(rcy)
```
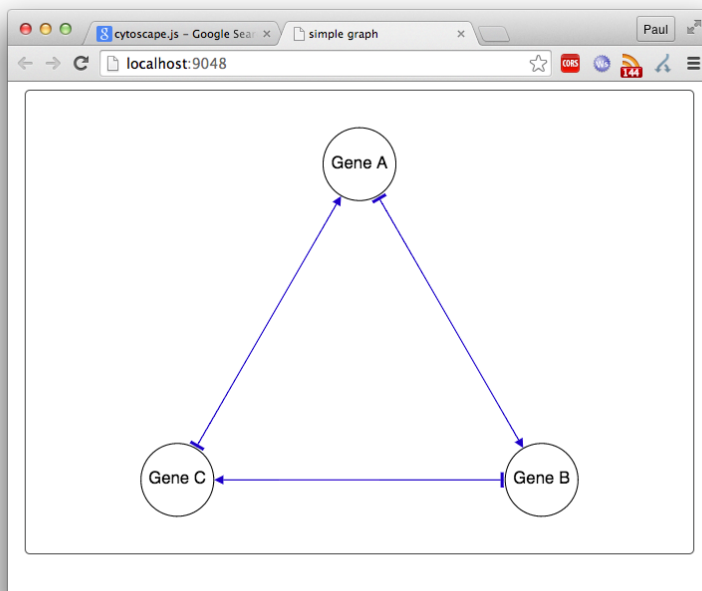

Node colors are perhaps the first part of the network you may want to render as a function of node attributes. We will use "lfc" (for log-fold change) frequently used as a measure of microarray expression ratios. We use the "interpolate" mode; intermediate values are interpolated from the control.points and their associated colors. We use *edgeType* to control edge color, in "lookup" mode. Note that not all rules have both modes. Node shape, for instance, offers only discrete options (ellipse, rectangle, etc.), and no mode need be specified.

```
> noa(g, "lfc")
> setNodeColorRule(rcy, "lfc", c(-5, 0, 5), c(green, white, red), mode="interpolate")
> setEdgeColorRule(rcy, "edgeType",
+                  c("phosphorylates", "synthetic lethal", "undefined"),
+                  c(red, green, blue), mode="lookup")
> redraw(rcy)
```
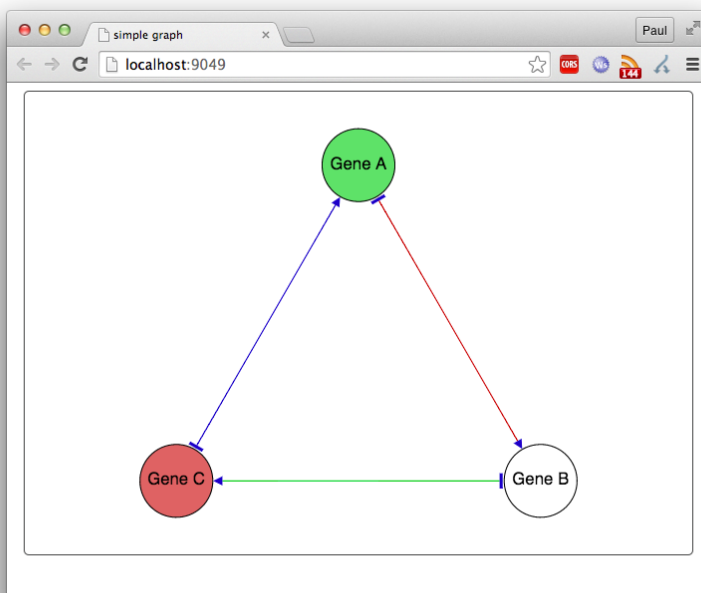

3   Larger Example: Hypoxia Induction Network

The *RefNet*) and *PSICQUIC* packages offer programmatic access to curated molecular (mostly protein) interactions. Among the interactions we have collected and offer through *RefNet* is one extracted from a Pouyssegur et al, 2006:

Nature, 2006 May 25;441(7092):437-43.

Hypoxia signalling in cancer and approaches to enforce tumour regression.

Pouysségur J[1], Dayan F, Mazure NM.

⊕ Author information

Abstract
Tumour cells emerge as a result of genetic alteration of signal circuitries promoting cell growth and survival, whereas their expansion relies on nutrient supply. Oxygen limitation is central in controlling neovascularization, glucose metabolism, survival and tumour spread. This pleiotropic action is orchestrated by hypoxia-inducible factor (HIF), which is a master transcriptional factor in nutrient stress signalling. Understanding the role of HIF in intracellular pH (pH(i)) regulation, metabolism, cell invasion, autophagy and cell death is crucial for developing novel anticancer therapies. There are new approaches to enforce necrotic cell death and tumour regression by targeting tumour metabolism and pH(i)-control systems.

PMID: 16724055 [PubMed - indexed for MEDLINE]

In this section, we load a *RefNet* data.frame describing the interactions reported in that paper, and display then using *RCyjs*. First we define a utility method for creating a *graphNEL* from a RefNet data.frame:

```
> refnetToGraphNEL <- function(tbl)
+ {
+   g = new ("graphNEL", edgemode="directed")
+   nodeDataDefaults(g, attr="type") <- "undefined"
+   nodeDataDefaults(g, attr="label") <- "default node label"
+   nodeDataDefaults(g, attr="expression") <-  1.0
+   nodeDataDefaults(g, attr="gistic") <-  0
+   nodeDataDefaults(g, attr="mutation") <-  "none"
+   edgeDataDefaults(g, attr="edgeType") <- "undefined"
+   edgeDataDefaults(g, attr= "pubmedID") <- ""
+   all.nodes <- sort(unique(c(tbl$A, tbl$B)))
+   g = graph::addNode (all.nodes, g)
+   nodeData (g, all.nodes, "label") = all.nodes
+   g = graph::addEdge (tbl$A, tbl$B, g)
+   edgeData (g, tbl$A, tbl$B, "edgeType") = tbl$type
+   g
+   }
```

Load *RefNet*, request the interactions described in the paper, convert them to a graphNEL:

```
> library(org.Hs.eg.db)
> library(RefNet)
> refnet <- RefNet();
> tbl.hypoxia <- interactions(refnet,provider="hypoxiaSignaling-2006")
> g.hypoxia <- refnetToGraphNEL(tbl.hypoxia)
```

Most of the the nodes in the graph are genes, and some are biological processes. The network display will be easier to grasp if these two categories are rendered differently. Use the *org.Hs.eg.db*: any node name mentioned there will be a gene, all others will be processes.

```
> all.nodes <- nodes(g.hypoxia)
> gene.nodes <- intersect(all.nodes, keys(org.Hs.egSYMBOL2EG))
> process.nodes <- setdiff(all.nodes, gene.nodes)
> nodeData(g.hypoxia, gene.nodes, attr="type") <- "gene"
> nodeData(g.hypoxia, process.nodes, attr="type") <- "process"
```

Create and display the network, using the first open port between 9047 and 9067 to connect to your default browser. Set some default rendering rules, and the browser window title. The nodes will be arranged randomly on the browser canvas.

```
> rcy <- RCyjs(portRange=9047:9067, quiet=TRUE, graph=g.hypoxia);
> setBackgroundColor(rcy, lightGray)
> setDefaultNodeSize(rcy, 60)
> setDefaultNodeColor(rcy, white)
> setDefaultNodeBorderColor(rcy, black)
> setDefaultNodeBorderWidth(rcy, 3)
> setNodeLabelRule(rcy, "label");
> setNodeShapeRule(rcy, "type", c("gene", "process"), c("ellipse", "roundrectangle"))
> redraw(rcy)
```

```
> title <- "Hypoxia Signaling: Pouyssegur 2006"
> setBrowserWindowTitle(rcy, title)
```

Cytoscape.js provides several automated layout algorithms, but optimal layout nearly always involves at least some manual placement of nodes as well. One such layout is included in the package. Recreate it by loading the file into which it was saved, and applying it. Then size the full network to occupy most of the browser canvas.

```
> saved.layout.file <- system.file(package="RCyjs", "extdata", "hypoxiaLayout.RData")
> stopifnot(file.exists(saved.layout.file))
> restoreLayout(rcy, saved.layout.file)
> fitContent(rcy)
> setZoom(rcy, 0.9 *getZoom(rcy))
```

Use the interaction type, specified in the *RefNet* data.frame, and copied into the graphNEL as the edge attribute "edgeType" to control the color of the displayed edges. Use this same information to control what shape, if any, is displayed to "decorate" the edge: a triangle indicates a positive effect of the upstream node on the downstream node; a "tee" indicates a negative effect. Finally, after specifying these rules, redraw the network and the rules are applied

```
> edgeColors <- list(activates = darkerGreen,
+                    inhibits = darkRed,
+                    inactivates = darkRed,
+                    hydroxylates = black,
+                    "TF cofactor" = darkBlue,
+                    "TF binding" = darkBlue,
+                    hydroxylated = black,
+                    stabilizes.mrna = darkerBlue,
+                    preserves = darkGreen,
+                    proteolyzes = darkRed)
> setEdgeColorRule(rcy, "edgeType", names(edgeColors), as.character(edgeColors), mode="lookup")
> setEdgeTargetArrowColorRule(rcy, "edgeType", names(edgeColors), as.character(edgeColors),
+                             mode="lookup")
> edgeTargetShapes <- list(activates = "triangle",
+                          inhibits = "tee",
+                          inactivates = "tee",
+                          hydroxylates = "triangle",
+                          "TF cofactor" = "none",
+                          "TF binding" = "triangle",
+                          hydroxylated = "none",
+                          stabilizes.mrna = "triangle",
+                          preserves = "triangle",
+                          proteolyzes = "triangle")
> setEdgeTargetArrowShapeRule(rcy, "edgeType", names(edgeTargetShapes), as.character(edgeTargetShapes))
> redraw(rcy)
>
>
```