Cancer networks and data

by Alexander Pico

2018-03-17

This vignette will demonstrate network retrieval from the STRING database, basic analysis, TCGA data loading and visualization in Cytoscape from R using the RCy3 package.

Installation

```
source("https://bioconductor.org/biocLite.R")
biocLite("RCy3")
library("RColorBrewer")
```

Required Software

The whole point of RCy3 is to connect with Cytoscape. You will need to install and launch Cytoscape:

- Download the latest Cytoscape from http://www.cytoscape.org/download.php
- Complete installation wizard
- Launch Cytoscape

For this vignette, you'll also need the STRING app to access the STRING database from within Cytoscape:

```
installApp('STRING')
```

Step 1: Getting Disease Networks

Use Cytoscape to query the STRING database for networks of genes associated with breast cancer and ovarian cancer.

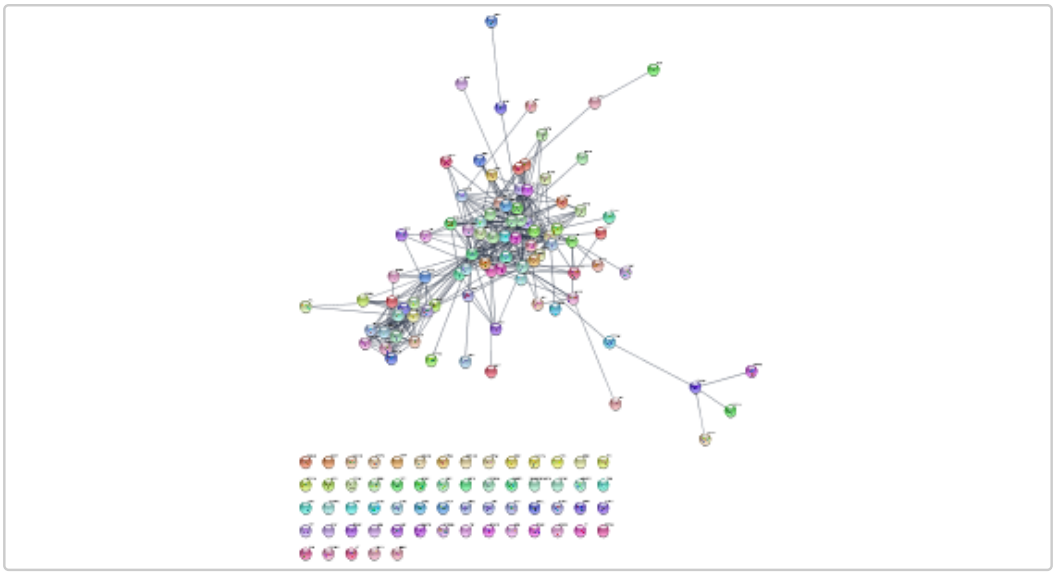
If the STRING app is not installed, no error is reported, but your network will be empty

Query STRING database by disease to generate networks

Breast cancer

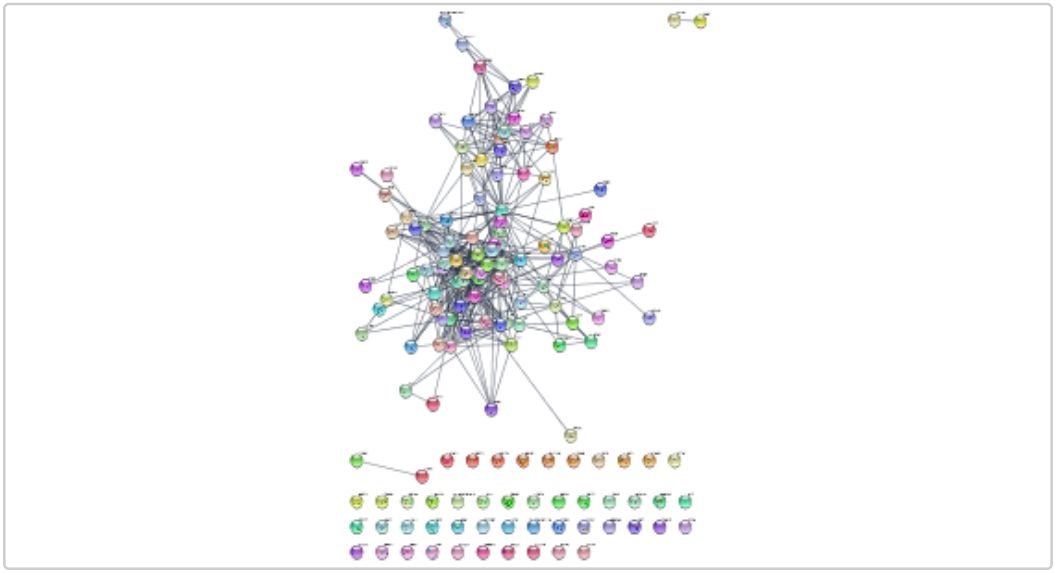
```
string.cmd = 'string disease query disease="breast cancer" cutoff=0.9 species="Homo sapiens"
limit=150'
commandsGET(string.cmd)
```

Here we are using Cytoscape's command line syntax, which can be used for any core or app automation function, and then making a GET request. Use *commandsHelp* to interrogate the functions and parameters available in your active Cytoscape session, including the apps you've installed!



Ovarian cancer

```
string.cmd = 'string disease query disease="ovarian cancer" cutoff=0.9 species="Homo sapiens"
limit=150'
commandsGET(string.cmd)
```



Step 2: Interacting with Cytoscape

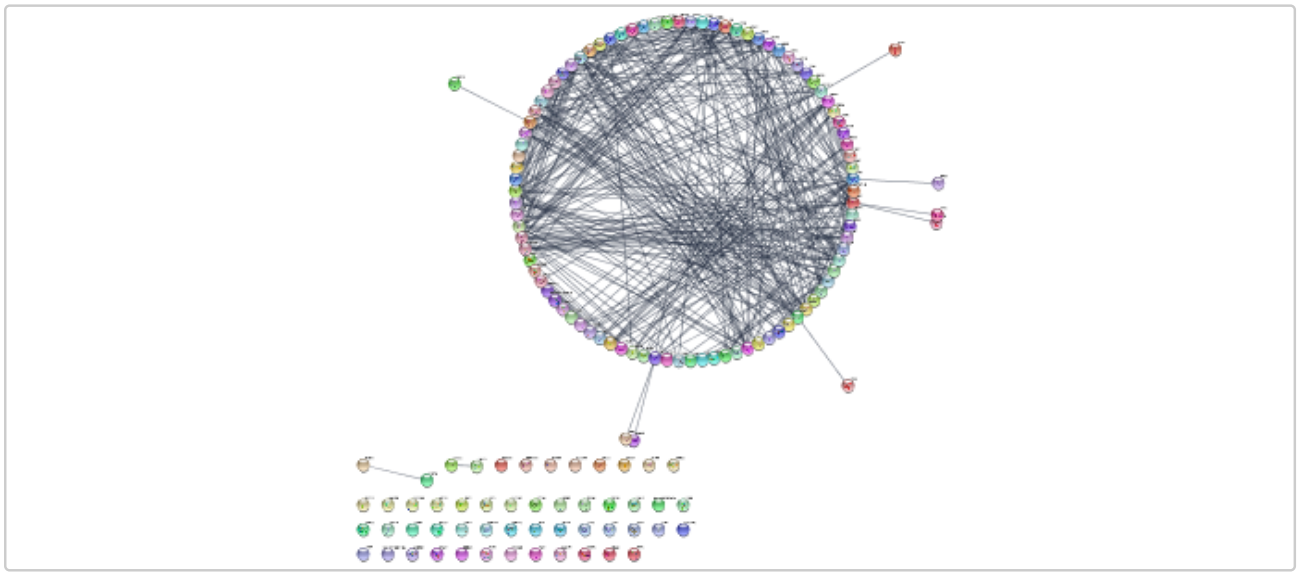
Now that we've got a couple networks into Cytoscape, let's see what we can go with them from R...

Get list of networks

```
getNetworkList()
```

Layout network

```
layoutNetwork(layout.name='circular')
```



List of layout algorithms available

```
getLayoutNames()
```

Layout with parameters!

```
getLayoutPropertyNames(layout.name='force-directed')
layoutNetwork('force-directed defaultSpringCoefficient=0.0000008 defaultSpringLength=70')
```

Get table data from network

Now, let's look at the tablular data associated with our STRING networks…

```
getTableColumnNames('node')
```

One of the great things about the STRING database is all the node and edge attriubtes they provide. Let's pull some of it into R to play with…

Retrieve disease scores

We can retrieve any set of columns from Cytoscape and store them as an R data frame.

```
disease.score.table <- getTableColumns('node','disease score')
disease.score.table
```

Plot distribution and pick threshold

Now you can use R like you normally would explore the data.

```
plot(factor(row.names(disease.score.table)),disease.score.table[,1],
ylab=colnames(disease.score.table)[1])
summary(disease.score.table)
```

Generate subnetworks

In order to reflect your exploration back onto the network, let's generate subnetworks…

…from top quartile of 'disease score'

```
top.quart <- quantile(disease.score.table[,1], 0.75)
top.nodes <- row.names(disease.score.table)[which(disease.score.table[,1]>top.quart)]
createSubnetwork(top.nodes,subnetwork.name ='top disease quartile')
#returns a Cytoscape network SUID
```

…of connected nodes only

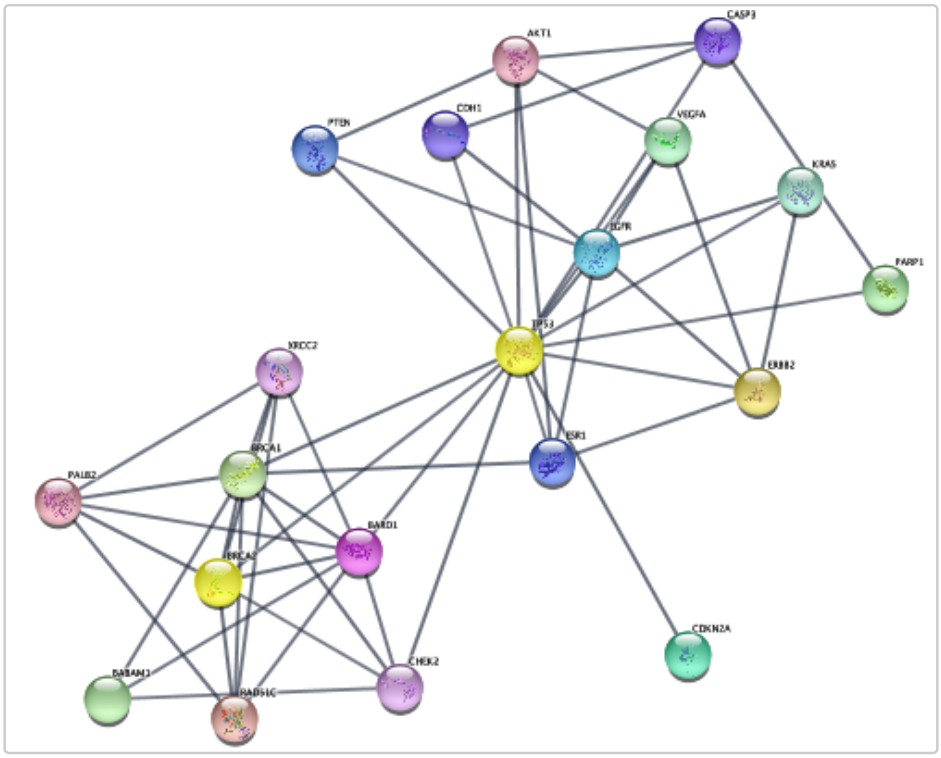
```
createSubnetwork(edges='all',subnetwork.name='top disease quartile connected')  #handy way to
exclude unconnected nodes!
```

…from first neighbors of top 3 genes, using the network connectivity together with the data to direct discovery.

```
setCurrentNetwork(network="String Network - ovarian cancer")
top.nodes <- row.names(disease.score.table)[tail(order(disease.score.table[,1]),3)]
selectNodes(nodes=top.nodes)
selectFirstNeighbors()
createSubnetwork('selected', subnetwork.name='top disease neighbors') # selected nodes, all
connecting edges (default)
```

…from diffusion algorithm starting with top 3 genes, using the network connectivity in a more subtle way than just first-degree neighbors.

```
setCurrentNetwork(network="String Network - ovarian cancer")
selectNodes(nodes=top.nodes)
commandsGET('diffusion diffuse') # diffusion!
createSubnetwork('selected',subnetwork.name = 'top disease diffusion')
layoutNetwork('force-directed')
```



Pro-tip: don't forget to **setCurrentNetwork()** the correct parent network before getting table column data and making selections.

Step 3: Visualizing data on networks

Load datasets

Downloaded TCGA data from https://portal.gdc.cancer.gov/ and preprocessed as R objects. Also available via each TCGA publication, e.g.:

- Breast: https://tcga-data.nci.nih.gov/docs/publications/brca_2012/
- Ovarian: https://tcga-data.nci.nih.gov/docs/publications/ov_2011/

```
load(system.file("extdata","tutorial-ovc-expr-mean-dataset.robj", package="RCy3"))
load(system.file("extdata","tutorial-ovc-mut-dataset.robj", package="RCy3"))
load(system.file("extdata","tutorial-brc-expr-mean-dataset.robj", package="RCy3"))
load(system.file("extdata","tutorial-brc-mut-dataset.robj", package="RCy3"))
```

Breast Cancer Datset

These datasets are similar to the data frames you normarlly encounter in R. For diversity, one using row.names to store corresponding gene names and the other uses the first column. Both are easy to import into Cytoscape.

```
str(brc.expr)  # gene names in row.names of data.frame
str(brc.mut)  # gene names in column named 'Hugo_Symbol'
```

Let's return to the Breast Cancer network…

```
setCurrentNetwork(network="String Network - breast cancer")
layoutNetwork('force-directed') #uses same settings as previously set
```

…and use the helper function from RCy3 called *loadTableData*

```
?loadTableData
loadTableData(brc.expr,table.key.column = "display name")  #default data.frame key is
row.names
loadTableData(brc.mut,'Hugo_Symbol',table.key.column = "display name")  #specify column name
if not default
```

Visual styles

Let's create a new style to visualize our imported data …starting with the basics, we will specify a few defaults and obvious mappings in a custom style all our own.

```
style.name = "dataStyle"
defaults.list <- list(NODE_SHAPE="ellipse",
                 NODE_SIZE=60,
                 NODE_FILL_COLOR="#AAAAAA",
                 EDGE_TRANSPARENCY=120)
node.label.map <- mapVisualProperty('node label','display name','p') # p for passthrough;
nothing else needed
createVisualStyle(style.name, defaults.list, list(node.label.map))
setVisualStyle(style.name=style.name)
```

Visualize expression data

Now let's update the style with a mapping for mean expression. The first step is to grab the column data from Cytoscape and pull out the min and max to define our data mapping range of values.

```
brc.expr.network = getTableColumns('node','expr.mean')
min.brc.expr = min(brc.expr.network[,1],na.rm=TRUE)
max.brc.expr = max(brc.expr.network[,1],na.rm=TRUE)
data.values = c(min.brc.expr,0,max.brc.expr)
```

Next, we use the RColorBrewer package to help us pick good colors to pair with our data values.

```
display.brewer.all(length(data.values), colorblindFriendly=TRUE, type="div") #
div,qual,seq,all
node.colors <- c(rev(brewer.pal(length(data.values), "RdBu")))
```

Finally, we use the handy *mapVisualProperty* function to construct the data object that CyREST needs to specify style mappings and then we'll send them off to Cytoscape with *updateStyleMappings*.

```
setNodeColorMapping('expr.mean', data.values, node.colors, style.name=style.name)
```

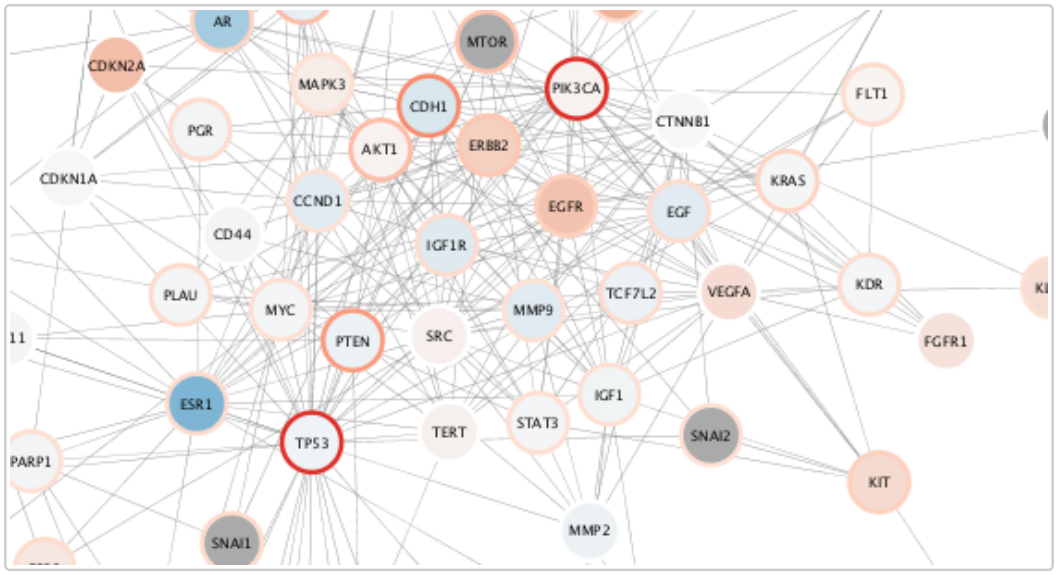
Pro-tip: depending on your data, it may be better to balance your color range over negative and positive values bounded by the largest min or max data value, so that color intensity scales similarly in both directions.

Visualize mutation data

OK, now let's update with a mapping for mutation. Here are all the same steps, but this time mapping mutation counts to *both* node border width and color.

```
brc.mut.network = getTableColumns('node','mut_count')
min.brc.mut = min(brc.mut.network[,1],na.rm=TRUE)
max.brc.mut = max(brc.mut.network[,1],na.rm=TRUE)
data.values = c(min.brc.mut,20,max.brc.mut)
display.brewer.all(length(data.values), colorblindFriendly=TRUE, type="seq")
border.colors <- c(brewer.pal(3, "Reds"))
setNodeBorderColorMapping('mut_count',data.values,border.colors,style.name=style.name)
border.width <- c(2,4,8)
setNodeBorderWidthMapping('mut_count',data.values,border.width,style.name=style.name)
```

This is a useful pair of visual properties to map to a single data column. See why?

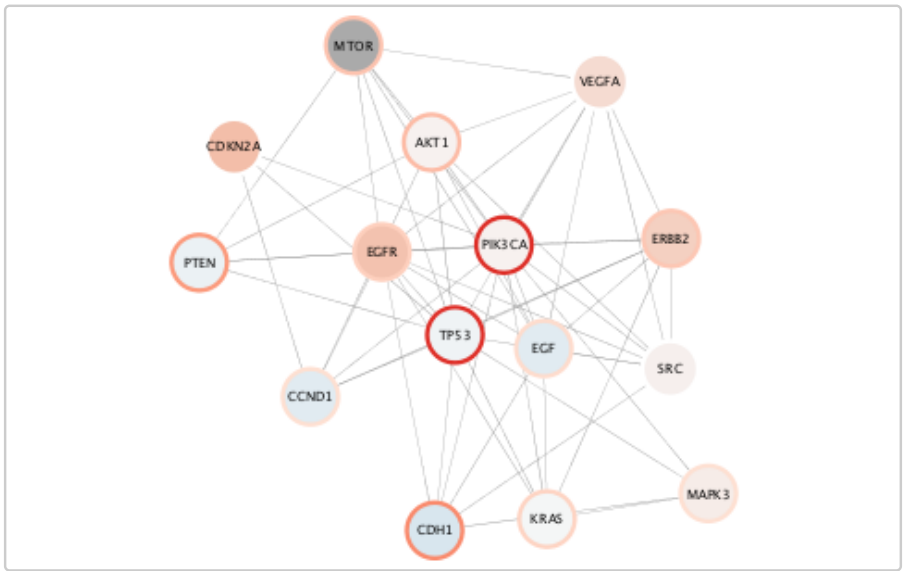


Subnetwork based on diffusion from heavily mutated nodes

Now, let's pull in what we learned about subnetwork selection and apply it here…

```
top.mut <- (brc.mut$Hugo_Symbol)[tail(order(brc.mut$mut_count),2)]
top.mut
selectNodes(nodes=top.mut,'display name')
commandsGET('diffusion diffuse')
createSubnetwork('selected',subnetwork.name = 'top mutated diffusion')
layoutNetwork('force-directed')
```

The top mutated genes are based on TCGA data and the diffusion algorithm is operating based on the network connectivity from STRING data, leading to a focused subnetwork view of critical Breast Cancer genes with mean patient expression data mapped to fill color. Now *that's* data integration!



Pro-tip: You can generate a legend for this in Cytoscape Style tab > Options > Create style… This is no yet available as a command. Coming soon!

Ovarian Cancer Datset

But what about the other network and datasets? Do we have to repeat *all* of those steps again? Actually, no!

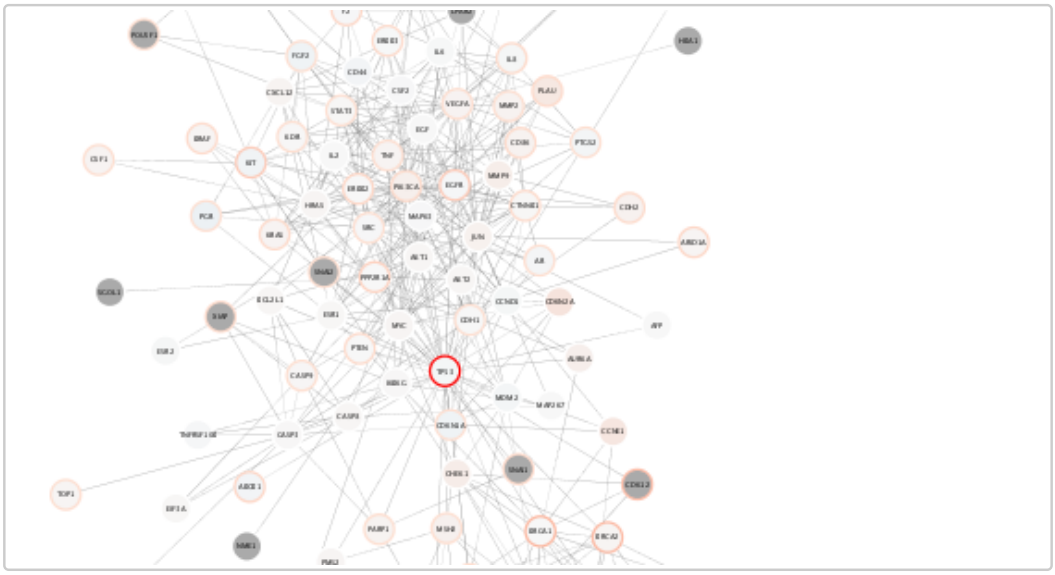
First, let's switch back over to the Ovarian Cancer network and load our data.

```
setCurrentNetwork(network="String Network - ovarian cancer")
clearSelection()
str(ovc.expr)  # gene names in row.names of data.frame
str(ovc.mut)  # gene names in column named 'Hugo_Symbol'

loadTableData(ovc.expr, table.key.column = 'display name')
loadTableData(ovc.mut,'Hugo_Symbol', table.key.column = 'display name')
```

Because we used the same column names in our original data frames, now we can simply apply the *same* visual style created above!

```
setVisualStyle(style.name=style.name)
```



Reusing the same style for both breast and ovarian cancers, we can compare the relative expression and mutation counts across the two datasets. For example, notice in the case of ovarian cancer: **decreased** range of mean expression and **fewer** mega-mutated genes.

Step 4: Saving, sharing and publishing

Saving a Cytoscape session file

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
```