We can now run our multi-city analysis by calling cityApply with fitSingleCity and the list of extractor functions in extractFun.

```
> results <- cityApply(fitSingleCity,
+     extractors = extractFun)
```

By default, cityApply applies the function specified in the FUN argument on all of the city dataframes in the currently registered database.

The effect estimates from the 4 cities can be pooled using a simple fixed effects model:

```
> beta <- sapply(results, "[[", "coef")
> std <- sapply(results, "[[", "std")
> weighted.mean(beta, 1/std^2) *
+     1000

[1] 0.2005406

> sqrt(1/sum(1/std^2)) * 1000

[1] 0.07230552
```

Future Directions

The NMMAPSdata package is a data package and we purposely omit any code for time series modeling. We are currently developing a separate package specifically designed for fitting time series models to air pollution and health data. For now, we hope that users will find the NMMAPSdata package useful for either reproducing results from previous studies or for implementing their own methods. Comments and suggestions are welcome.

Bibliography

M. J. Daniels, F. Dominici, S. L. Zeger, and J. M. Samet. *The National Morbidity, Mortality, and Air Pollution Study, Part III: Concentration-Response Curves and Thresholds for the 20 Largest US Cities*. Health Effects Institute, Cambridge MA, 2004. 10

F. Dominici, M. Daniels, S. L. Zeger, and J. M. Samet. Air pollution and mortality: Estimating regional and national dose-response relationships. *Journal of the American Statistical Association*, 97:100–111, 2002a. 11

F. Dominici, A. McDermott, M. Daniels, S. L. Zeger, and J. M. Samet. Mortality among residents of 90 cities. In *Revised Analyses of Time-Series Studies of Air Pollution and Health*, pages 9–24. The Health Effects Institute, Cambridge MA, 2003. 10, 11

F. Dominici, A. McDermott, S. L. Zeger, and J. M. Samet. On the use of generalized additive models in time-series studies of air pollution and health. *American Journal of Epidemiology*, 156(3):193–203, 2002b. 11

R. D. Peng, L. J. Welty, and A. McDermott. The National Morbidity, Mortality, and Air Pollution Study database in R. Technical Report 44, Johns Hopkins University Department of Biostatistics, 2004. http://www.bepress.com/jhubiostat/paper44/. 10

J. M. Samet, F. Dominici, S. L. Zeger, J. Schwartz, and D. W. Dockery. *The National Morbidity, Mortality, and Air Pollution Study, Part I: Methods and Methodological Issues*. Health Effects Institute, Cambridge MA, 2000a. 10

J. M. Samet, S. L. Zeger, F. Dominici, F. Curriero, I. Coursac, D. W. Dockery, J. Schwartz, and A. Zanobetti. *The National Morbidity, Mortality, and Air Pollution Study, Part II: Morbidity and Mortality from Air Pollution in the United States*. Health Effects Institute, Cambridge MA., 2000b. 10

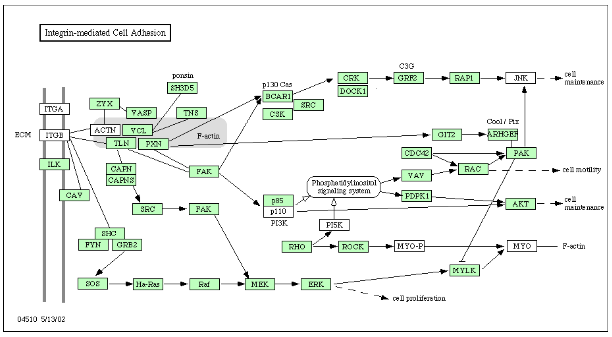
Laying Out Pathways With Rgraphviz

by Jeff Gentry, Vincent Carey, Emden Gansner and Robert Gentleman

Overview

Graphviz http://www.graphviz.org is a flexible tool for laying out and rendering graphs. We have developed an R interface to the Graphviz functionality. In this article we demonstrate the use of *Rgraphviz* to layout molecular pathways, but note that the tool is much more general and can be used to layout any graph.

In this article, we will use the *hsa041510* pathway from KEGG (http://www.genome.ad.jp/kegg/pathway/hsa/hsa04510.html), which is available as a *graph* object from the *graph* package as the *integrinMediatedCellAdhesion* dataset. This dataset contains the graph as well as a list of attributes that can be used for plotting. The pathway graph as rendered by KEGG is seen here:



Obtaining the initial graph

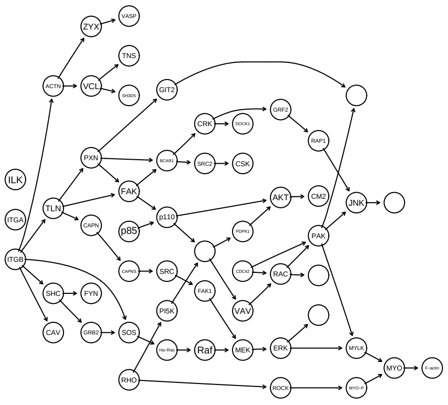
At this time, there is no automated way to extract the appropriate information from KEGG (or other sites) and construct a graph. If one wishes to layout their own pathways, it requires manual construction of a graph, creating each node and then recording the edges. Likewise, for any basic attributes (such as the green/white coloration in the hsa041510 graph), they too must be collected by hand. For instance, this would be a good time to take advantage of edge weights by putting in desired values (which can be changed later, if necessary) while constructing the edges of the graph. We have manipulated some of the weights, such as the weight between the p85 and p110 nodes, as they are intended to be directly next to each other. Once constructed, the graph can be saved with the save command and stored for later use (which has been done already as part of the *integrinMediatedCellAdhesion* dataset).

```
> library("Rgraphviz")

Loading required package: graph
Loading required package: cluster
Loading required package: Ruuid
Creating a new generic function for "print" in
"Ruuid"
Loading required package: Biobase
Welcome to Bioconductor
      Vignettes contain introductory material.
      To view, simply type: openVignette()
      For details on reading vignettes, see
      the openVignette help page.

> data("integrinMediatedCellAdhesion")
> IMCAGraph

A graph with  directed  edges
Number of Nodes = 55
Number of Edges = 62
```

Laying out the graph

Laying out a pathway graph is much like dealing with any other graph, except that typically we want to as closely emulate the officially laid out graph (or at least make it look like an actual pathway - the Graphviz layout methods were not designed with this task in mind). A lot of experimentation comes into play, in order to find the right combination of attributes, although there are some general tips that can help out. The first thing to know is that we will almost always want to use the *dot* layout, as that will provide the closest base to work off. Likewise, the *rankdir* attribute should be set to *LR*, to give us the left to right look of the graph. To see our starting point, here is the *IMCAGraph* with just those settings.

```
> plot(IMCAGraph,
+       attrs = list(graph =
+          list(rankdir = "LR")))
```



Note that *IMCAAttrs$defAttrs* is simply the *rankdir* attribute for *graph*, so we will be using that in place of the `list` call from now on.

This plot is not terrible, it does convey the proper information, but the layout is quite different from the layout at KEGG, and can be difficult to interpret. Furthermore, smaller things like the coloration of the nodes and the shape of the phosphatidylinositol signaling system are not handled.

Using other attributes can have a positive effect. We can set the color of each node (this must be entered manually) and change the shape of the phosphyatidylinositol signaling system node to be an ellipse. We have done this for this graph in the *IMCAAttrs$nodeAttrs* data:

```
> IMCAAttrs$nodeAttrs$shape

Phosphatidylinositol signaling system
                            "ellipse"

> IMCAAttrs$nodeAttrs$fillcolor[1:10]
```

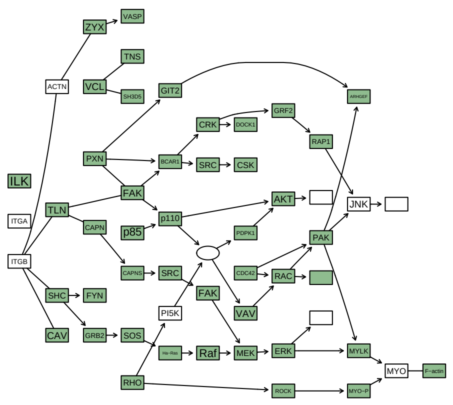
```
  ITGB                                ITGA
  "white"                             "white"
  ACTN                                JNK
  "white"                             "white"
  MYO                                 MYOP
  "white"                             "white"
  PI5K  Phosphatidylinositol signaling system
  "white"                             "white"
  cell maintenance                    CM2
  "white"                             "white"
```

We have set up a few other attributes. You'll notice on the original plot that there are some nodes that have the same label, there are two *cell maintenance* nodes, 2 *FAK* nodes, and 2 *SRC* nodes. In the internal structure of the graph we have given these nodes different names but we set their labels to be the same as the original. Also, we have defined some edges that do not exist in the original graph for structural reasons and make their color transparent so that they are not displayed. We also change some of the arrowheads:

```
> IMCAAttrs$nodeAttrs$label

             CM2                 FAK1
"cell maintenance"              "FAK"
             SRC2
            "SRC"

> IMCAAttrs$edgeAttrs$color

     ITGB~SOS       ACTN~VCL       TLN~PXN
"transparent" "transparent" "transparent"

> IMCAAttrs$edgeAttrs$arrowhead

 ITGB~ACTN   ITGB~TLN   ACTN~ZYX    VCL~TNS
    "none"     "none"     "none"     "none"
 VCL~SH3D5   TLN~CAPN    TLN~FAK  PAK~ARGHEF
    "none"     "none"     "none"     "none"
  PXN~FAK    ITGB~CAV   ITGB~SHC  MYO~F-actin
    "none"     "none"     "none"     "none"
```

Using these attributes to plot the graph will get us a bit closer to our goal:

```
> plot(IMCAGraph, attrs = IMCAAttrs$defAttrs,
+      nodeAttrs = IMCAAttrs$nodeAttrs,
+      edgeAttrs = IMCAAttrs$edgeAttrs)
```



Now the color scheme is the same as KEGG and using an ellipse helps with the rendering of the phosphatidylinositol signaling system node. However, we're still left with the issue that the layout itself is not the same as the original and is harder to interpret. The output nodes are scattered, there is no clear sense of where the membrane nodes are, and many nodes that are intended to be close to each other are not. This is where the use of subgraphs and clusters can help. In Graphviz, a subgraph is an organizational method to note that a set of nodes and edges belong in the same conceptual space, and share attributes. Subgraphs have no impact on layouts themselves, but are used to group elements of the graph for assigning attributes. A Graphviz cluster is a subgraph which is laid out as a separate graph and then introduced into the main graph. This provides a mechanism for clustering the nodes in the layout. For a description of how to specify subgraphs in *Rgraphviz*, please see the vignette `HowTo Render A Graph Using Rgraphviz` from the *Rgraphviz* package.

Here we define four subgraphs: One will be the membrane nodes, one will be the output nodes, one will be the F-actin block and the last will be the combination of the *PAK*, *JNK* and *ARHGEF* nodes to get the verticle stacking. It would be possible to specify more subgraphs to try to help keep things more blocked together like the original graph, but for the purposes of this document, these are what will be used.

```
> sg1 <- subGraph(c("ILK", "ITGA",
                  "ITGB"), IMCAGraph)
> sg2 <- subGraph(c("cell maintenance",
+                 "cell motility",
+                 "cell proliferation",
+                 "F-actin"), IMCAGraph)
> sg3 <- subGraph(c("ACTN", "VCL", "TLN",
+                "PXN"), IMCAGraph)
```

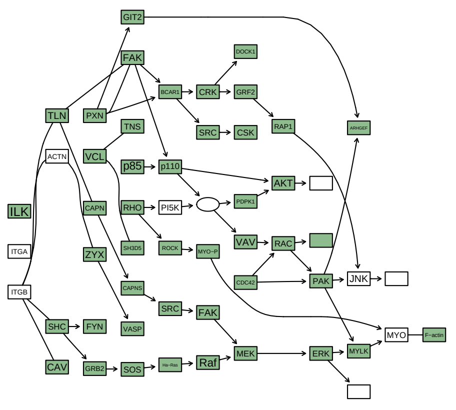
```
> sg4 <- subGraph(c("PAK", "JNK", "ARHGEF"),
+                 IMCAGraph)
```

We have defined the subgraphs. We can use these as subgraphs or clusters in Graphviz. Ideally, we would like to use clusters, as that guarantees that the nodes will be laid out close together. However, we want to use the *rank* attribute for the membrane, output nodes and the *ARHGEF* block, specifically using the values *min* and *max* and *same*, respectively. That will help with the verticle alignment that we see in the KEGG graph and create more of the left to right orientation. The problem is that *rank* currently only works with subgraphs and not clusters. So for these three subgraphs, we will be defining them as Graphviz subgraphs, and the F-actin block will be defined as a cluster. We have already prepared all of this as *IMCAAttrs$subGList*:

```
> IMCAAttrs$subGList

[[1]]
[[1]]$graph
A graph with  undirected  edges
Number of Nodes = 3
Number of Edges = 0

[[1]]$cluster
[1] FALSE

[[1]]$attrs
[[1]]$attrs$rank
[1] "min"

[[2]]
[[2]]$graph
A graph with  undirected  edges
Number of Nodes = 4
Number of Edges = 0

[[2]]$cluster
[1] FALSE

[[2]]$attrs
[[2]]$attrs$rank
[1] "max"

[[3]]
[[3]]$graph
A graph with  undirected  edges
Number of Nodes = 4
Number of Edges = 1

[[4]]
[[4]]$graph
A graph with  undirected  edges
Number of Nodes = 3
Number of Edges = 1

[[4]]$cluster
[1] FALSE

[[4]]$attrs
[[4]]$attrs$rank
[1] "same"
```

You can see that we have set the *rank* attribute on the three subgraphs and that the F-actin subgraph has been defined as a cluster. Using this subgraph list, we now get:

```
> plot(IMCAGraph, attrs = IMCAAttrs$defAttrs,
+      nodeAttrs = IMCAAttrs$nodeAttrs,
+      edgeAttrs = IMCAAttrs$edgeAttrs,
+      subGList = IMCAAttrs$subGList)
```

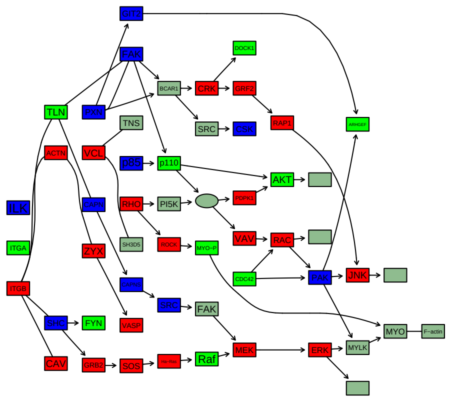


While this is still not identical to the image on KEGG (and for most graphs, it will be hard to do so), this layout is now easier to interpret. We can see the output nodes are now to the right side of the graph, and the membrane nodes are stacked on the left of the graph. We can also see the F-actin group in the upper left portion of the graph, representing the cluster.

Working with the layout

One of the benefits of using *Rgraphviz* to perform your layout as opposed to using the static layouts provided by sites like KEGG, is the ability to work with outside data and visualize it using your graph. The `plotExpressionGraph` function in *geneplotter* can be used to take expression data and then color nodes based on the level of expression. By default, this function will color nodes blue, green or red, corresponding to expression levels of 0-100, 101-500, and 501+ respectively. Here we will use this function along with the *fibroEset* and *hgu95av2* data packages and the *IMCAAttrs$IMCALocuLink* data which maps the nodes to their LocusLink ID values.

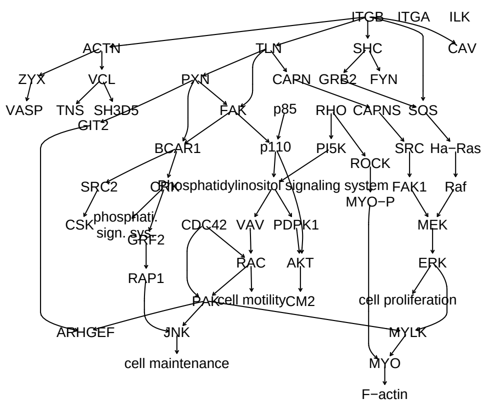
```
> require("geneplotter")

Loading required package: geneplotter
Loading required package: annotate
Loading required package: reposTools
[1] TRUE

> require("fibroEset")

Loading required package: fibroEset
[1] TRUE

> require("hgu95av2")

Loading required package: hgu95av2
[1] TRUE

> data("fibroEset")
> plotExpressionGraph(IMCAGraph,
+                     IMCAAttrs$LocusLink,
+                     exprs(fibroEset)[,1],
+                     hgu95av2LOCUSID,
+                     attrs =
+                     IMCAAttrs$defAttrs,
+                     subGList =
+                     IMCAAttrs$subGList,
+                     nodeAttrs =
+                     IMCAAttrs$nodeAttrs,
+                     edgeAttrs =
+                     IMCAAttrs$edgeAttrs)
```



One can also simply choose to layout the pathway based on the needs and desires of a particular situation. For instance, the following layout could be used in situations where the node names are the important visual cue, as opposed to the previous example where the nodes themselves are being used to demonstrate values:

```
> z <- IMCAGraph
> nodes(z)[39] <- c("phosphati.\nsign. sys.")
```

```
> nag <- agopen(z, name = "nag",
+               attrs = list(node =
+               list(color = "white", fontcolor =
+               "white"), edge =
+               list(arrowsize = 2.8, minlen = 3)))
> nagxy <- getNodeXY(nag)

> plot(nag)
> text(nagxy, label = nodes(z), cex = 0.8)
```



Conclusions

Rgraphviz provides a flexible interface to Graphviz to obtain layout information. Rendering the graph is handled in R, using R graphics. There are still a few rough edges but the package is quite flexible and can be used to layout and render any graph. Graph layout is still very much an art. You will seldom get a good layout without doing some tweaking and adjusting. We have demonstrated a few of the very many tools that Graphviz offers.

Jeff Gentry
DFCI
jgentry@jimmy.harvard.edu

Vincent Carey
Channing Lab
stvjc@channing.harvard.edu

Emden Gansner
AT&T Labs
erg@research.att.com

Robert Gentleman
DFCI
rgentlem@jimmy.harvard.edu