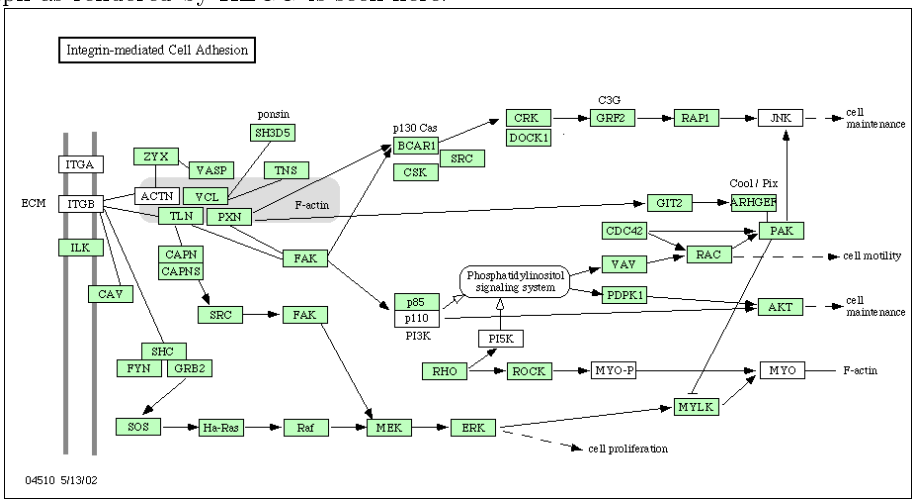
HowTo layout a pathway

Jeff Gentry

April 25, 2023

1 Overview

This article will demonstrate how you can use *Rgraphviz* to layout and render pathways, such as the ones available at KEGG (`http://www.genome.ad.jp/kegg/pathway/`). For demonstration purposes, we will be working with 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:



2 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).

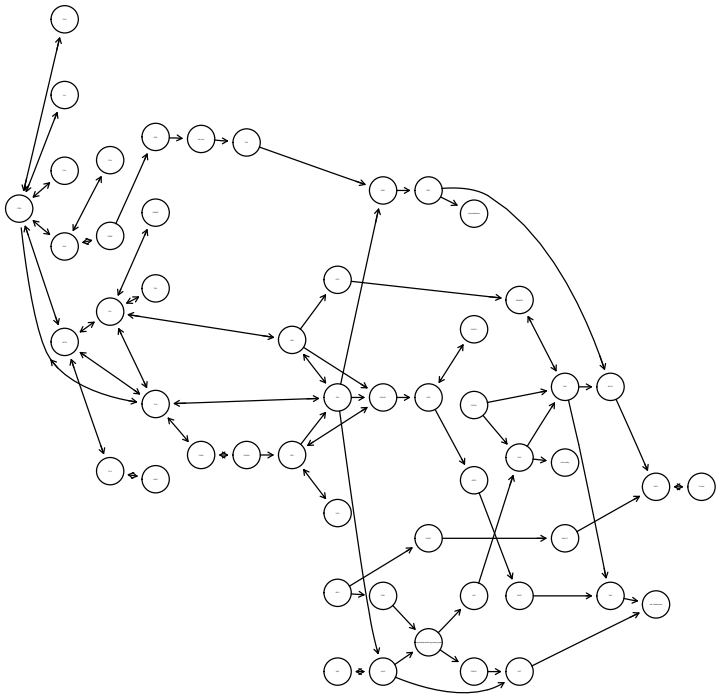
```
> library("Rgraphviz")
> data("integrinMediatedCellAdhesion")
> IMCAGraph

A graphNEL graph with directed edges
Number of Nodes = 52
Number of Edges = 91
```

3 Laying out the graph

Laying out a pathway graph is much like dealing with any other graph, except that typically we want to 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 experimenation 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. We will use the *attrs* list to store all the layout parameters as we move on.

```
> attrs <- list(graph=list(rankdir="LR"))
> IMCAGraph <- layoutGraph(IMCAGraph, attrs=attrs)
> renderGraph(IMCAGraph)
```



This plot is not terrible, in that it conveys the proper information, but the formatting is quite different from the layout at KEGG, and can be difficult to get a coherent idea of what is going on. Furthermore, smaller things like the coloration of the nodes and the shape of the phosphatidylinositol signaling system are not being represented here. Here is where using other attributes can start to have a positive effect. Let us first start with the node labels. The default behavior of `renderGraph` is to compute a common font size for all node labels in a way that they all fit their node. There are some long node names in the graph, and we can determine the length of each node name using the function `nchar`.

```
> n <- nodes(IMCAGraph)
> names(labels) <- labels <- n
> nc <- nchar(labels)
> table(nc)

nc
 3  4  5  6  7 13 16 18 37
25 13  7  2  1  1  1  1  1

> long <- labels[order(nc, decreasing=TRUE)][1:4]
> long

  Phosphatidylinositol signaling system                    cell proliferation
"Phosphatidylinositol signaling system"                  "cell proliferation"
```

```
                    cell maintenance                          cell motility
                  "cell maintenance"                        "cell motility"
```

We need to deal with these four long names separately. One option would be to use an alternative name, maybe an abbreviation. Alternatively, we could include line feeds into the strings in order to force multi-line text. This is what we do. The escape sequence for line feeds in R is `\n`.

```
> labels[long] <- c(paste("Phosphatidyl-\ninositol\n",
+     "signaling\nsystem", sep=""), "cell\nproliferation",
+     "cell\nmaintenance", "cell\nmotility")
```

Because we want to change a property of individual nodes we have to use `nodeRenderInfo` for the setting. The function matches rendering parameters by the name of the list item and nodes by the names of the items of the individual vectors. The parameter we want to modify is *label*.

```
> nodeRenderInfo(IMCAGraph) <- list(label=labels)
> renderGraph(IMCAGraph)
```

The four labels are now plotted as multi-line strings but this has not changed the layout. Remember that rendering and layout are two distinct processes, and for changes to affect the latter you have to re-run `layoutGraph`. Another layout change we may want to do at this point is to further increase the size of the nodes with long names to give them a little bit more room for the labels. Also, we do not want a fixed size for all the nodes but rather allow *Graphviz* to adapt the node size to fit the labels. This is controlled by the logical layout parameter *fixedsize*.

```
> attrs$node <- list(fixedsize=FALSE)
> width <- c(2.5, 1.5, 1.5, 1.5)
> height <- c(1.5, 1.5, 1.5, 1.5)
> names(width) <- names(height) <- long
> nodeAttrs <- list(width=width, height=height)
> IMCAGraph <- layoutGraph(IMCAGraph, attrs=attrs,
+                          nodeAttrs=nodeAttrs)
> renderGraph(IMCAGraph)
```

It also makes sense to use a rectangular shape for all but the "Phosphatidylinositol signaling system" node which actually comprises a fairly substantial cellular subprocess and we want it to be highlighted accordingly. The best way to do that is to set the shape argument using the `nodeRenderInfo` function. We can use "rectangle" as the default and set the "Phosphatidylinositol signaling system" node to "ellipse". The other three nodes with the long names and also the "F-actin" node represent processes rather than physical objects and we do not want to plot shapes for them, but display plain text of the node names instead (Figure ??).

```
> shape <- rep("rectangle", length(n))
> names(shape) <- n
> shape[long[1]] <- "ellipse"
> shape[c(long[2:4], "F-actin")] <- "plaintext"
> nodeRenderInfo(IMCAGraph) <- list(shape=shape)
> IMCAGraph <- layoutGraph(IMCAGraph, attrs=attrs,
+                          nodeAttrs=nodeAttrs)
> renderGraph(IMCAGraph)
```

What is still missing in our graph is some color. Looking at Figure 1 we can see that there seem to be different classes of nodes, some colored green and others remaining transparent, and we want to reproduce this color scheme for our plot.

```
> colors <- rep("lightgreen", length(n))
> names(colors) <- n
> transp <- c("ITGB", "ITGA", "MYO", "ACTN", "JNK", "p110",
+             "Phosphatidylinositol signaling system",
+             "PI5K", "MYO-P", "cell maintenance", "cell motility",
+             "F-actin", "cell proliferation")
> colors[transp] <- "transparent"
> nodeRenderInfo(IMCAGraph) <- list(fill=colors)
> renderGraph(IMCAGraph)
```

Here the color scheme is now the same as on 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 doesn't convey the same meaning as the original. The output nodes are scattered about, there's not a clear sense of where the membrane nodes are, and many nodes that are intended to be close to each other simply 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, sharing attributes and the like. While there is some tendency to have nodes be laid out near each other in a subgraph, there is no guarantee of this, and the results can be highly dependent on the layout method (*dot*, *neato*, etc). A Graphviz cluster is a subgraph which is laid out as a separate graph and then introduced into the main graph. This provides a much stronger guarantee of having the nodes clustered together visually. For a description of how to specify subgraphs in *Rgraphviz*, please see the vignette `HowTo Render A Graph Using Rgraphviz`.

So here we will define four subgraphs: One will be the membrane nodes, one will be the output nodes, one will be the cytoskeleton components and the last will be everything else. 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("ITGA", "ITGB", "ILK", "CAV"), IMCAGraph)
> sg2 <- subGraph(c("cell maintenance", "cell motility",
+                   "F-actin", "cell proliferation"), IMCAGraph)
```

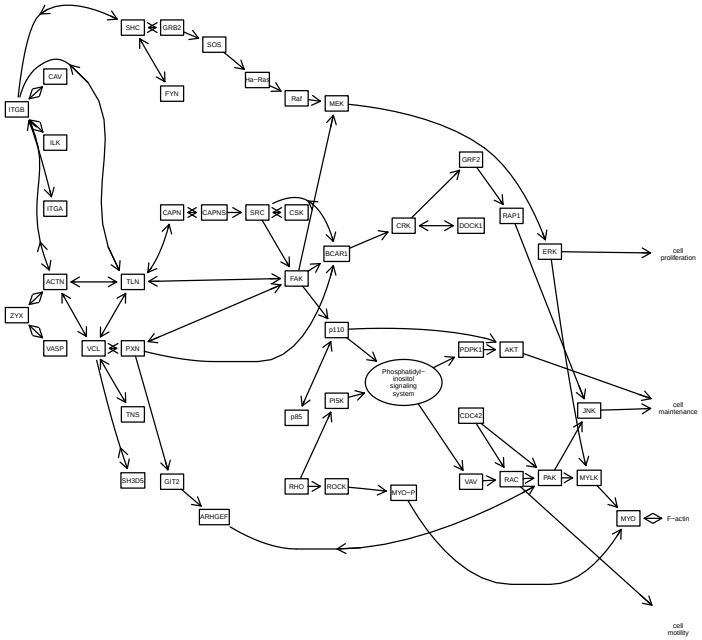
```
> sg3 <- subGraph(c("ACTN", "VCL", "TLN", "PXN", "TNS", "VASP"),
+                 IMCAGraph)
> sg4 <- subGraph(setdiff(n, c(nodes(sg1), nodes(sg2), nodes(sg3))),
+                 IMCAGraph)
```

While we have the subgraphs defined, we still have not determined whether to 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, it would also be useful to utilize the *rank* attribute for the membrane and output nodes, specifically using the values *source* and *sink* respectively. That will help to get the verticle line up that we see in the KEGG graph and create more of the left to right pathway feel. The problem is that *rank* only works with subgraphs and not clusters. So for the membrane and output subgraphs, we will be defining them as Graphviz subgraphs, and the other two subgraphs will be defined as clusters:

```
> subGList <- vector(mode="list", length=4)
> subGList[[1]] <- list(graph=sg1, attrs=c(rank="source"))
> subGList[[2]] <- list(graph=sg2, attrs=c(rank="sink"))
> subGList[[3]] <- list(graph=sg3, cluster=TRUE)
> subGList[[4]] <- list(graph=sg3, cluster=TRUE)
```

You can see that subgraphs 1 and 3 have the *cluster* parameter set to *FALSE* as well as having a *rank* attribute set appropriate. Subgraphs 2 and 4 simply have the subgraph itself, and will be laid out as a cluster without any special attributes. Using this subgraph list, we now get:

```
> IMCAGraph <- layoutGraph(IMCAGraph, attrs=attrs,
+                          nodeAttrs=nodeAttrs, subGList=subGList)
> renderGraph(IMCAGraph)
```



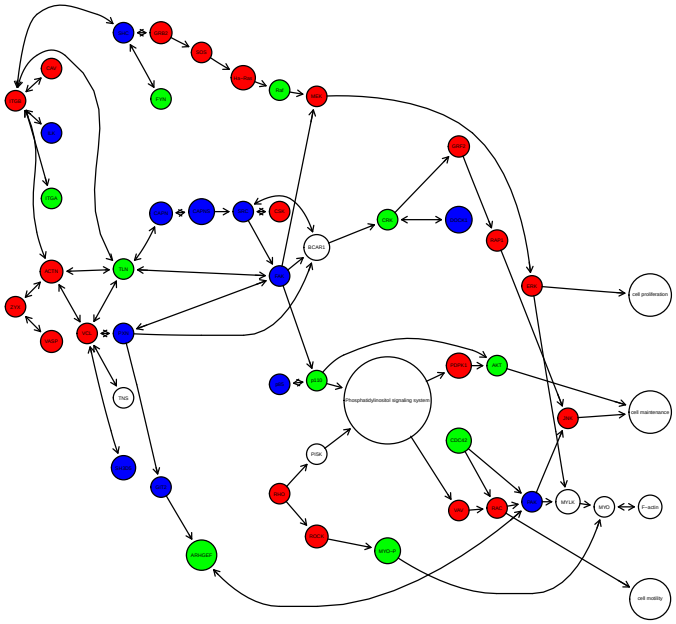
While this is still not identical to the image on KEGG (and for most graphs, it will be impossible given current abilities to do so), this layout is now much closer to providing an accurate visual rendition of the pathway. We can see the output nodes are now to the right end of the graph, and while not neatly stacked on the left hand side the membrane nodes are to the left side of the rest. We can also see the F-actin group in the lower left portion of the graph, representing one of the clusters.

4 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.db* data packages and the *IMCAAttrs$IMCALocuLink* data which maps the nodes to their LocusLink ID values.

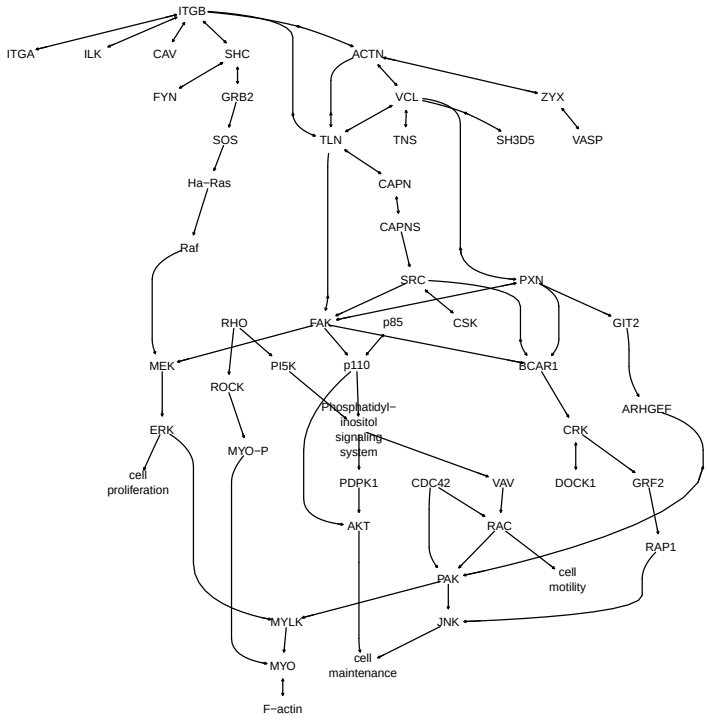
```
> require("geneplotter")
> require("fibroEset")
> require("hgu95av2.db")
```

```
> data("fibroEset")
> plotExpressionGraph(IMCAGraph, IMCAAttrs$LocusLink,
+                     exprs(fibroEset)[,1], hgu95av2ENTREZID,
+                     attrs=attrs,
+                     subGList=subGList, nodeAttr=nodeAttrs)
```



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
> nodeRenderInfo(z) <- list(shape="plaintext", fontsize=100)
> nag <- layoutGraph(z, attrs=list(edge=list(arrowsize=2.8, minlen=3)))
> renderGraph(nag)
```



5 Conclusions

At this time, laying out a pathway can provide good visual information for users, although it isn't yet able to be completely automated nor is it a perfect science. Yet with a bit of work and experimentation, one can get a fairly close rendition of what is available on sites like KEGG and have the ability to directly manipulate the graphs and customize the outputs to demonstrate a variety of effects. Hopefully as time goes on, we will be able to provide more in the way of automation in our tools, but even as it exists now, laying out pathways can provide a valuable tool for users.

6 Sessioninfo

This document was produced using

```
> sessionInfo()
```

```
R version 4.3.0 RC (2023-04-13 r84269 ucrt)
Platform: x86_64-w64-mingw32/x64 (64-bit)
Running under: Windows Server 2022 x64 (build 20348)
```

```
Matrix products: default


locale:
[1] LC_COLLATE=C
[2] LC_CTYPE=English_United States.utf8
[3] LC_MONETARY=English_United States.utf8
[4] LC_NUMERIC=C
[5] LC_TIME=English_United States.utf8

time zone: America/New_York
tzcode source: internal

attached base packages:
[1] stats4    grid      stats     graphics  grDevices utils     datasets
[8] methods   base

other attached packages:
 [1] hgu95av2.db_3.13.0   org.Hs.eg.db_3.17.0  fibroEset_1.41.0
 [4] geneplotter_1.78.0   annotate_1.78.0      XML_3.99-0.14
 [7] AnnotationDbi_1.62.0 IRanges_2.34.0       S4Vectors_0.38.0
[10] lattice_0.21-8       Biobase_2.60.0       biocGraph_1.62.0
[13] Rgraphviz_2.44.0     graph_1.78.0         BiocGenerics_0.46.0

loaded via a namespace (and not attached):
 [1] bit_4.0.5               compiler_4.3.0          crayon_1.5.2
 [4] blob_1.2.4              bitops_1.0-7            Biostrings_2.68.0
 [7] png_0.1-8               fastmap_1.1.1           R6_2.5.1
[10] XVector_0.40.0          GenomeInfoDb_1.36.0     GenomeInfoDbData_1.2.10
[13] DBI_1.1.3               RColorBrewer_1.1-3      rlang_1.1.0
[16] KEGGREST_1.40.0         cachem_1.0.7            bit64_4.0.5
[19] RSQLite_2.3.1           memoise_2.0.1           cli_3.6.1
[22] zlibbioc_1.46.0         xtable_1.8-4            vctrs_0.6.2
[25] RCurl_1.98-1.12         httr_1.4.5              pkgconfig_2.0.3
[28] tools_4.3.0
```

together with the following version of graphviz

```
> graphvizVersion()

$installed_version
[1] '2.28.0'

$build_version
[1] '2.28.0'
```

```
$bundled_graphviz
[1] TRUE
```