HowTo Render A Graph Using Rgraphviz

Jeff Gentry

November 25, 2003

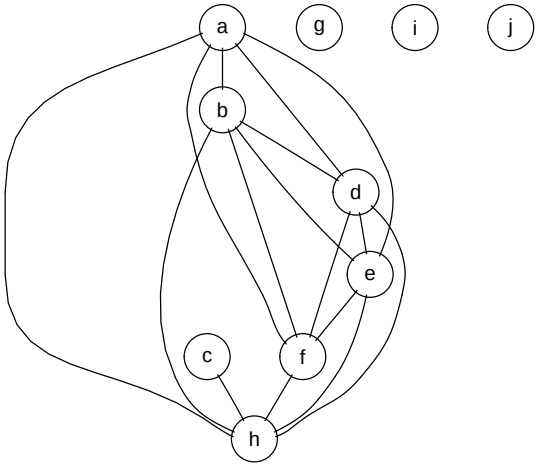
1 Overview

This article will demonstrate how to easily render a graph from R into various formats using the *Rgraphviz*. To do this, first we need to generate a R graph using the *graph* package:

```
> library(Rgraphviz)

Creating a new generic function for "lines" in ".GlobalEnv"
Creating a new generic function for "plot" in ".GlobalEnv"

> set.seed(123)
> V <- letters[1:10]
> M <- 1:4
> g1 <- randomGraph(V, M, 0.2)
> g1

A graph with  undirected  edges
Number of Nodes = 10
Number of Edges = 16
```

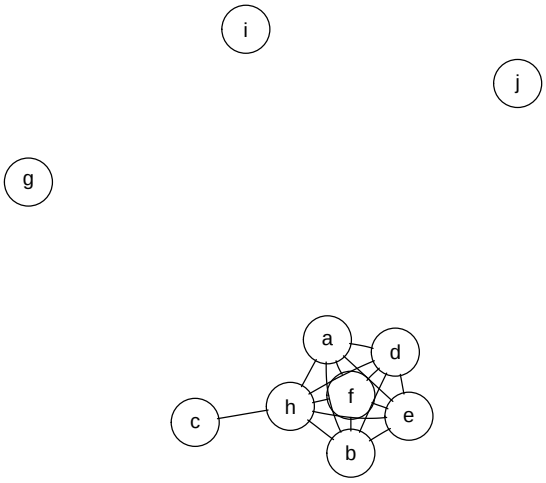
2 Plotting in R Using Different Layout Methods

It is quite simple to generate a R plot window to display your graph. Once you have your graph object, simply use the `plot` method:



The *Rgraphviz* package allows you to specify varying layout engines, such as ”dot” (the default), ”neato”, and ”twopi”. This can be done using the call to `plot`:

```
> z <- plot(g1, "neato")
```

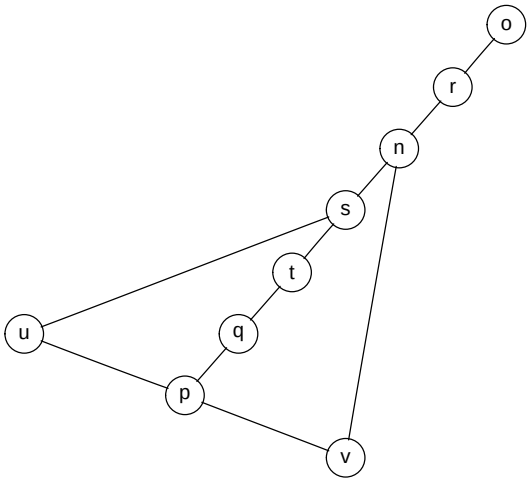


The "twopi" layout method requires a graph to be fully connected. To determine if your graph is fully connected:

```
> isConnected(g1)

[1] FALSE
```

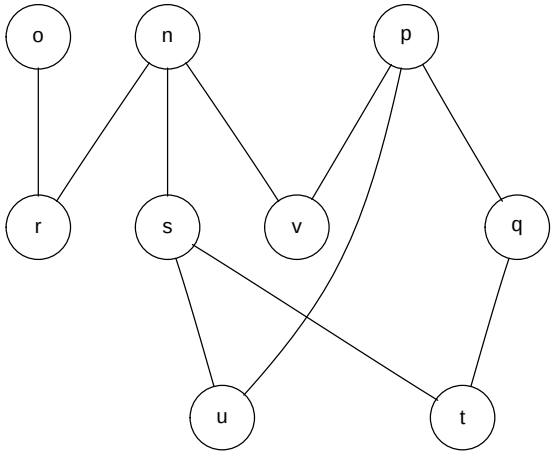
A working "twopi" layout can be seen with this graph:

```
> set.seed(123)
> V <- letters[14:22]
> g2 <- randomEGraph(V, 0.2)
> isConnected(g2)

[1] TRUE

> z <- plot(g2, "twopi")
```

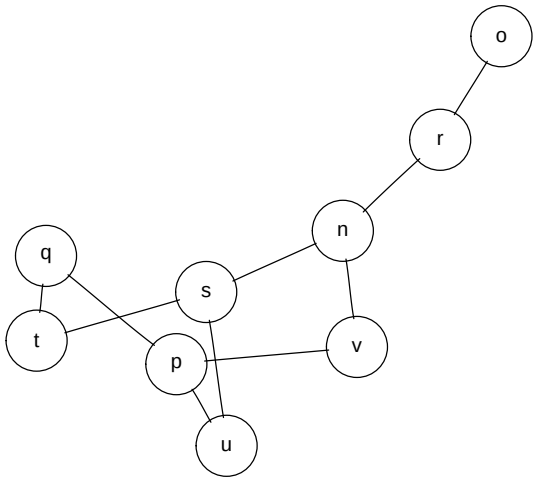


And finally, to demonstrate how the differing layout methods work on this second graph:

```
> z <- plot(g2, "dot")
```




```
> z <- plot(g2, "neato")
```



3 SubGraphs

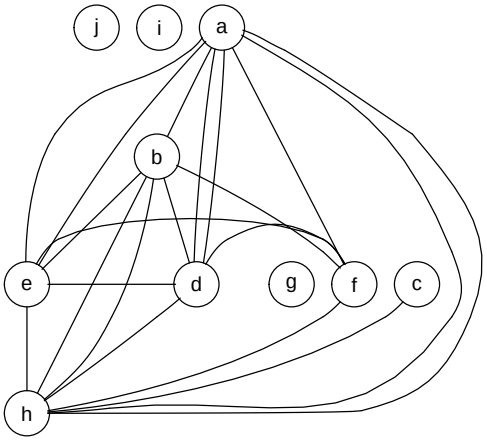
Rgraphviz supports the ability to define specific clustering of nodes. This will instruct the layout algorithm to attempt to keep the clustered nodes close together. To do this, one must first generate the desired set (one or more) of subgraphs with the *graph* object.

```
> sg1 <- subGraph(c("a", "d", "j", "i"), g1)
> sg1

A graph with  undirected  edges
Number of Nodes = 4
Number of Edges = 1

> sg2 <- subGraph(c("b", "e", "h"), g1)
> sg2

A graph with  undirected  edges
Number of Nodes = 3
Number of Edges = 3

> sg3 <- subGraph(c("c", "f", "g"), g1)
> sg3
```

```
A graph with  undirected  edges
Number of Nodes = 3
Number of Edges = 0
```

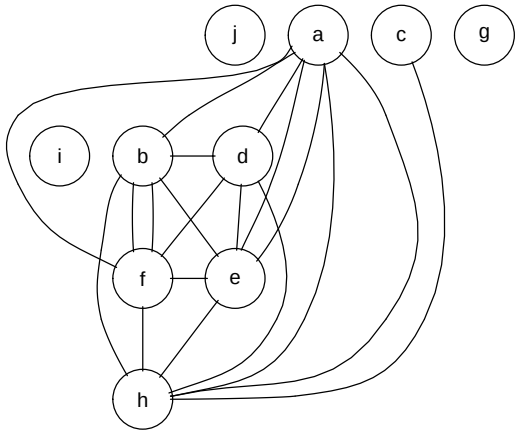
To plot using the subgraphs, one must use the `subGList` argument which accepts a list of every subgraph.

```
> plot(g1, subGList = list(sg1, sg2, sg3))
```



To demonstrate the differences that will appear with different subgraph patterns, another example is provided:

```
> sg1 <- subGraph(c("a", "c", "d", "e", "j"), g1)
> sg2 <- subGraph(c("f", "h", "i"), g1)
> plot(g1, subGList = list(sg1, sg2))
```



4 Labels

Users can specify both node and edge labels to the plot. Node labels are provided using the argument "nodeLabels" and consist of a character vector which must be equal in length to the number of nodes in the graph. The first node label in the vector corresponds to the first node, etc. If no nodeLabels argument is provided, the graph will simply use the node names as labels.

Edge labels are slightly more complex, and are represented as a list. This list elements correspond to the 'tail' ('from') nodes. Each element contains a named character vector where the elements of the vector are the edge labels and the names of said vector are the 'head' ('to') node that corresponds with that edge. As an example:

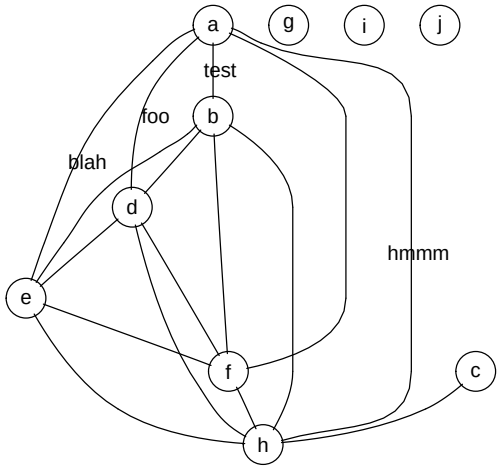
In both situations, the user can alternatively supply a single value to either "nodeLabels" or "edgeLabels", and in that case that value will be used for all of the node or edge labels (as appropriate for the parameter).

```
> edgeLabs <- vector(length = length(nodes(g1)), mode = "list")
> names(edgeLabs) <- names(edges(g1))
> for (i in 1:10) edgeLabs[[i]] <- vector(mode = "character")
> aLabs <- c("test", "foo", "blah", "hmmm")
> names(aLabs) <- c("b", "d", "e", "h")
```

```
> edgeLabs$a <- aLabs
> edgeLabs$a

     b      d      e      h
"test"  "foo" "blah" "hmmm"
```

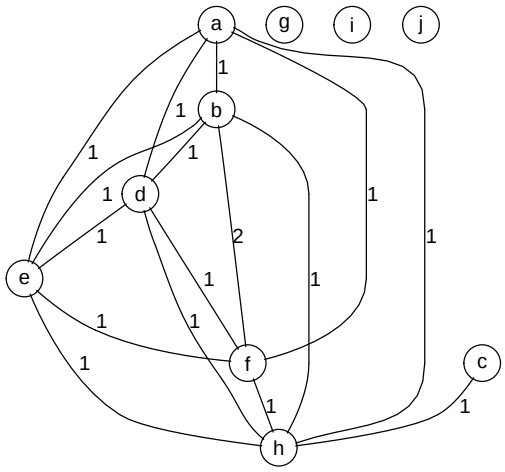
This set up the initial empty list and then filled in labels for each edge connecting to node 'a' in this graph. These labels can then be used in the plot:

```
> plot(g1, edgeLabels = edgeLabs)
```



For users who wish to use the edge weights as the labels, there is a convenience method `weightLabels` provided which will provide the same information as `edgeWeights` but in the appropriate format for the `edgeLabels` parameter. As an example:

```
> plot(g1, edgeLabels = weightLabels(g1))
```



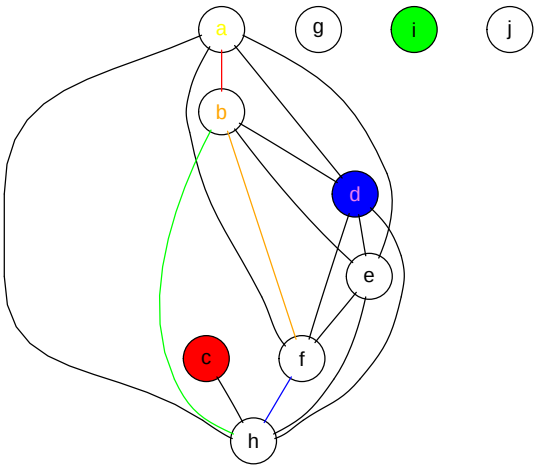
5 Adding Some Color

Many aspects of the plotted graph can be colorized to help highlight certain features. Nodes, node labels and edges can all have special coloring within a plot. In all three cases there is a default color provided by the system, and a user can specifically note deviations from that default. The default colors are defined by the arguments "defNodeCol", "defTextCol" and "defEdgeCol". Any color can be specified to any of these, although suitable defaults are chosen if nothing is specified.

Specifying deviations from the default color is also easy. The respective arguments here are "nodeCols", "textCols" and "edgeCols". In the former two cases, the system takes a named vector - and any element whose name matches the name of a node will have the specified color used for that node. Edges are a bit more complex, and are represented as a list of node names (the tail nodes), each of which contain a list of node names (the head nodes) which contain a color. For every edge, if an element in the list exist x*tail*head, that color will be used for the edge.

```
> nodeCols <- c("red", "blue", "green")
> names(nodeCols) <- c("c", "d", "i")
> textCols <- c("yellow", "orange", "violet")
```

```
> names(textCols) <- c("a", "b", "d")
> edgeCols <- list()
> edgeCols$a$b <- "red"
> edgeCols$f$h <- "blue"
> edgeCols$b$h <- "green"
> edgeCols$b$f <- "orange"
> plot(g1, nodeCols = nodeCols, textCols = textCols, edgeCols = edgeCols)
```



6 The Attributes List

There are many visualization options in Graphviz that can be set beyond those which are given explicit options using Rgraphviz. The user can manually specify values for any of these by specificing an attribute list which will get passed down to Graphviz. An example of an attribute that users would commonly want to specify is "dir" which notes the direction of a directed edge - for instance the default is "forward", but one could set "backwards", "none" (ie it is visually undirected) or even "both" (so one can represent an edge in each direction between two nodes using only one actual edge and not two). A list of all available attributes is accessible online at: http://www.research.att.com/ erg/graphviz/info/attrs.html. (note that not all of these will have a visible effect in Rgraphviz)

To manually specify attributes, one must first create an attributes list, which

is a list of length three with the names "graph", "node", and "edge" (corresponding to graph attributes, node attributes and edge attributes respectively). Each one of these list elements is itself a list of the appropriate attributes. For instance, to define the "dir" attribute to be "both" as mentioned above:

```
> attrs <- vector(length = 3, mode = "list")
> names(attrs) <- c("graph", "node", "edge")
> attrs$edge$dir <- "both"
> attrs

$graph
NULL

$node
NULL

$edge
$edge$dir
[1] "both"
```

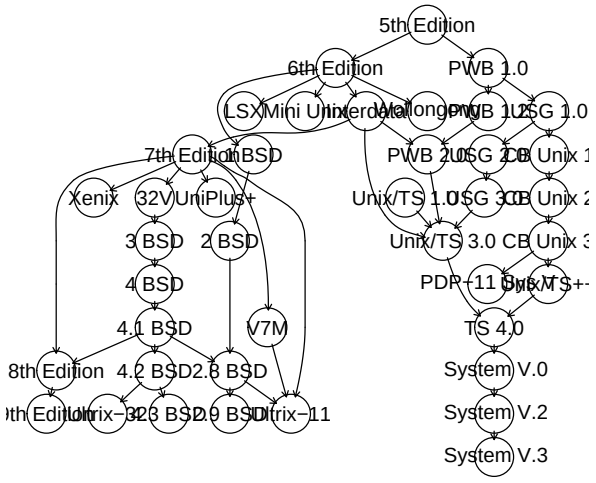
7 Example Graphs

Here are some other examples of graphs that can be drawn:

Unix history, using "dot" and "neato" layouts ...

```
> z <- new("graphNEL", nodes = c("5th Edition", "6th Edition",
+     "Interdata", "7th Edition", "V7M", "8th Edition", "1 BSD",
+     "2 BSD", "2.8 BSD", "32V", "3 BSD", "4 BSD", "4.1 BSD", "4.2 BSD",
+     "PWB 1.0", "USG 1.0", "CB Unix 1", "CB Unix 2", "CB Unix 3",
+     "USG 2.0", "USG 3.0", "PWB 2.0", "Unix/TS 1.0", "Unix/TS 3.0",
+     "Unix/TS++", "TS 4.0", "System V.0", "System V.2", "LSX",
+     "Mini Unix", "Wollongong", "Ultrix-11", "Xenix", "UniPlus+",
+     "9th Edition", "2.9 BSD", "Ultrix-32", "PDP-11 Sys V", "System V.3",
+     "4.3 BSD", "PWB 1.2"), edgeL = list("5th Edition" = list(edges = c(2,
+     15)), "6th Edition" = list(edges = c(29, 7, 30, 31, 3)),
+     Interdata = list(edges = c(24, 22, 4)), "7th Edition" = list(edges = c(6,
+         10, 5, 32, 33, 34)), V7M = list(edges = 32), "8th Edition" = list(edges = 35),
+     "1 BSD" = list(edges = 8), "2 BSD" = list(edges = 9), "2.8 BSD" = list(edges = c(32,
+         36)), "32V" = list(edges = 11), "3 BSD" = list(edges = 12),
+     "4 BSD" = list(edges = 13), "4.1 BSD" = list(edges = c(14,
+         9, 6)), "4.2 BSD" = list(edges = c(40, 37)), "PWB 1.0" = list(edges = c(41,
+         16)), "PWB 1.2" = list(edges = 22), "USG 1.0" = list(edges = c(17,
+         20)), "CB Unix 1" = list(edges = 18), "CB Unix 2" = list(edges = 19),
+     "CB Unix 3" = list(edges = c(25, 38)), "USG 2.0" = list(edges = 21),
+     "USG 3.0" = list(edges = 24), "PWB 2.0" = list(edges = 24),
+     "Unix/TS 1.0" = list(edges = 24), "Unix/TS 3.0" = list(edges = 26),
```

```
+     "Unix/TS++" = list(edges = 26), "TS 4.0" = list(edges = 27),
+     "System V.0" = list(edges = 28), "System V.2" = list(edges = 39),
+     LSX = character(), "Mini Unix" = character(), Wollongong = character(),
+     "Ultrix-11" = character(), Xenix = character(), "UniPlus+" = character(),
+     "9th Edition" = character(), "2.9 BSD" = character(), "Ultrix-32" = character(),
+     "PDP-11 Sys V" = character(), "System V.3" = character(),
+     "4.3 BSD" = character()), edgemode = "directed")
> a <- plot(z, nodeCols = c("orange", "yellow"))
> a <- plot(z, "neato", nodeCols = c("lightblue", "yellow", "lightgreen",
+     "orange"))
```



OS process map, using ”twopi” layout ... z <- new(”graphNEL”, nodes=c(”run”,”intr”,”runbl”,”kernel”,”zombi
edgeL=list(”run”=list(edges=c(2,3,4)),”intr”=list(edges=c(1,3)), ”runbl”=list(edges=c(1,2)),
”kernel”=list(edges=c(1,5,6,7)), ”zombie”=list(edges=4), ”sleep”=list(edges=c(4,7,8)),
”runmem”=list(edges=c(4,6,9,10)), ”swap”=list(edges=c(6,9)), ”runswap”=list(edges=c(7,8,10)),
”new”=list(edges=c(7,9)))) a <- plot(z, ”twopi”)