A New Interface to Render Graphs Using Rgraphviz

Florian Hahne

April 17, 2009

Contents

1 Overview

This vignette shows how to use Rgraphviz's updated interface for rendering of graphs. For details on graph layout see the Vignette "How To Plot A Graph Using Rgraphviz". Note that the design of the interface is independent of graphviz, however no bindings to any other graph layout software have been implemented so far.

2 Introduction

There are two distinct processes when plotting graphs: *layout*, which places nodes and edges in a (usually two-dimensional) space, and *rendering*, which is the actual drawing of the graph on a graphics device. The first process is typically the more computationally expensive one and relies on sophisticated algorithms that arrange the graph's components based on different criteria. The arrangement of the nodes and edges depends on various parameters such as the desired node size, which again may be a function of the size of the node labels.

Rendering of a graph is often subject to frequent changes and adaptions, and it makes sense to separate the two processes in the software implementation. It is also important to realize that the process of getting a good layout is iterative, and using default parameter settings seldom yields good plots.

The code available for doing graph layout in Bioconductor is based mainly on the *Graphviz* project and the *Boost graph library*. However, because the rendering of a graph is separated from the layout, one can use other graph layout algorithms, as long as the requirements of the rendering interface are met.

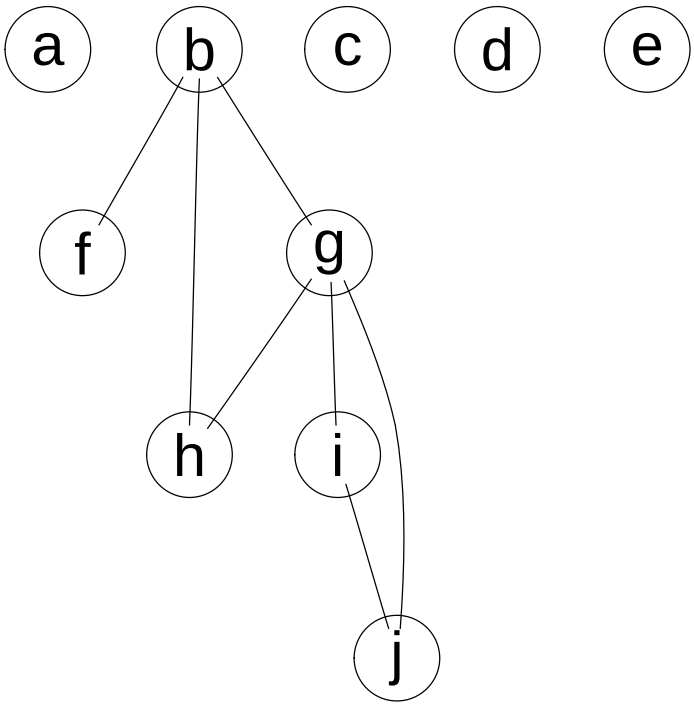
In the process of laying out a graph some amount of information is generated, mostly regarding the locations and dimensions of nodes on a two-dimensional plane and the trajectories of the edges. Bioconductor *graph* objects now contain a slot `renderInfo` to hold this information. The typical workflow of a graph layout is to pass a graph object to the layout function, which returns another graph object containing all the necessary information for subsequent rendering. The process of calling a layout algorithm is encapsulated in the `layoutGraph` function. Calling this function without any further arguments will result in using one of the *Graphviz* layout algorithms via the the *Rgraphviz* package. We assume a knowledge of graph layout and the available *Graphviz* options in the remainder of this Vignette and will only deal with the rendering part, here.

The rendering of a graph relies solely on *R*'s internal plotting capabilities. As for all other plotting functions in *R*, many parameters controlling the graphical output can be tuned. However, because there are several parts of a graph one might want to modify (e.g., nodes, edges, captions), setting the graphical parameters is slightly more complex than for other plots. We have established a hierarchy to set global defaults, graph-specific parameters, and settings that apply only to individual rendering operations.

To demonstrate the new rendering interface we first generate a graph using the *graph* package and lay it out using the default *Graphviz* dot layout.

```
> library("Rgraphviz")
> set.seed(123)
> V <- letters[1:10]
> M <- 1:4

> g1 <- randomGraph(V, M, 0.2)

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

> g1 <- layoutGraph(g1)

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

> renderGraph(g1)
```

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



3 Default rendering parameters

There is a hierarchy to set rendering parameters for a graph. The levels of this hierarchy are

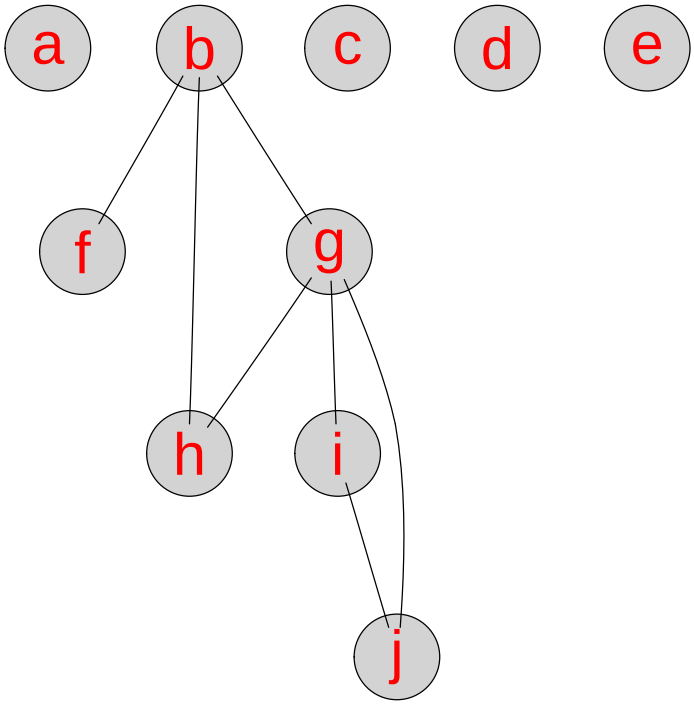
1. The session: These are the defaults that will be used for a parameter if not set somewhere further down the hierarchy. You can change the session defaults at any time using the function `graph.par`.
2. Rendering operation: Defaults can be set for a single rendering operation, that is, a call to `renderGraph` using its `graph.pars` argument.
3. Individual nodes or edges: Parameters for individual nodes or edges can be set using the `nodeRenderInfo` and `edgeRenderInfo` functions.

3.1 Default node parameters

We now use our example graph to further explore these options. Let's start with the nodes: We want to fill all our nodes with a gray color and use a red color

for the node names. Since this should be applied to all nodes, we set a global rendering parameter using `graph.par`:

```
> graph.par(list(nodes = list(fill = "lightgray", textCol = "red")))
> renderGraph(g1)
```



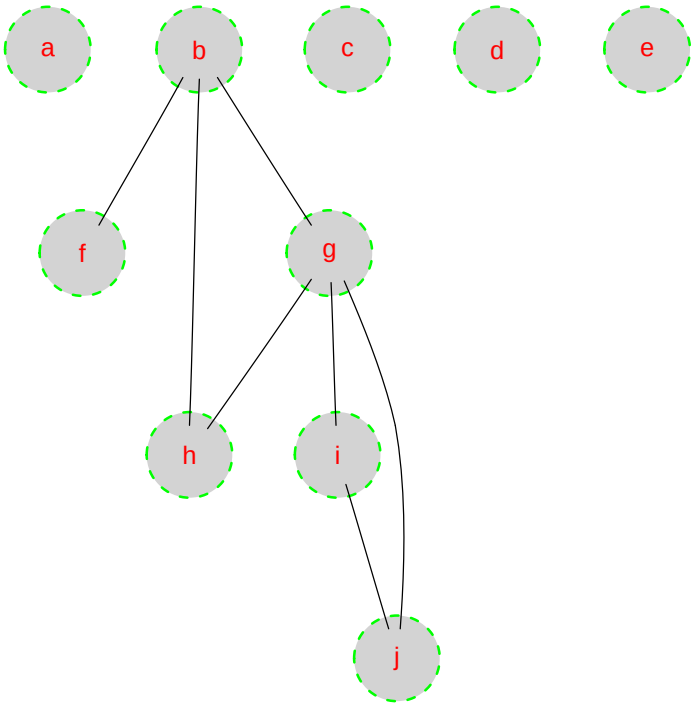
Note that `graph.par` takes as single argument a list of rendering parameters. There are three different types of parameters the user might want to set: nodewide parameters, edgewide parameters and parameters that control features of the whole graph. Accordingly, the parameters list passed to `graph.par` may contain the list items `nodes`, `edges` and `graph`. Each of these list items can again be a list, now of parameters. In our example, the parameters are `fill` and `textCol`. All currently available node parameters are:

- col: the color of the line drawn as node border. Defaults to `black`.
- lty: the type of the line drawn as node border. Defaults to `solid`. Valid values are the same as for the R's base graphic parameter `lty`.
- lwd: the width of the line drawn as node border. Defaults to `1`.
- fill: the color used to fill a node. Defaults to `transparent`.
- textCol:the font color used for the node labels. Defaults to `black`.

- fontsize: the font size for the node labels in points. Defaults to `14`. Note that the fontsize will be automatically adjusted to make sure that all labels fit their respective nodes. You may want to increase the node size by supplying the appropriate layout parameters to *Graphviz* in order to allow for larger fontsizes.
- cex: Expansion factor to further control the fontsize. As default, this parameter is not set, in which case the fontsize will be clipped to the node size. This mainly exists to for consistency with the base graphic parameters and to override the clipping of fontsize to nodesize.

In the next code chunk we set the defaults for all remaining node parameters:

```
> graph.par(list(nodes = list(col = "green", lty = "dashed", lwd = 2,
+     fontsize = 6)))
> renderGraph(g1)
```



Similar to *R*'s base `par` function, the original values of a modified paramter are returned by `graph.par` and you may want to assign them to an object in order to revert them later.

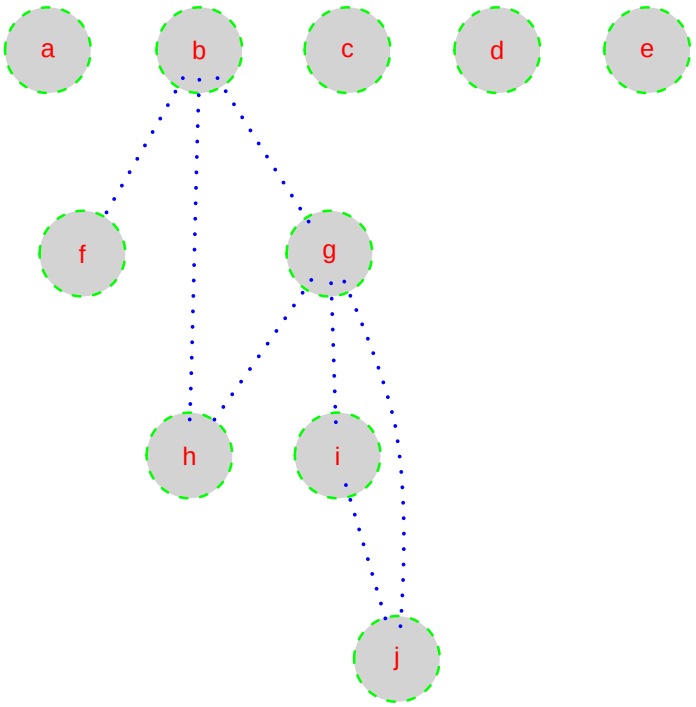
3.2 Default edge parameters

Now, let's take a look at the parameters that control the appearence of the edges. They are:

- col: the color of the edge line. Defaults to `black`.
- lty: the type of the edge line. Defaults to `solid`. Valid values are the same as for the R's base graphic parameter `lty`.
- lwd: the width of the edge line. Defaults to `1`.
- textCol:the font color used for the edge labels. Defaults to `black`.
- fontsize: the font size for the edge labels in points. Defaults to `14`.
- cex: Expansion factor to further control the fontsize. This mainly exists to be consistent with the base graphic parameters.

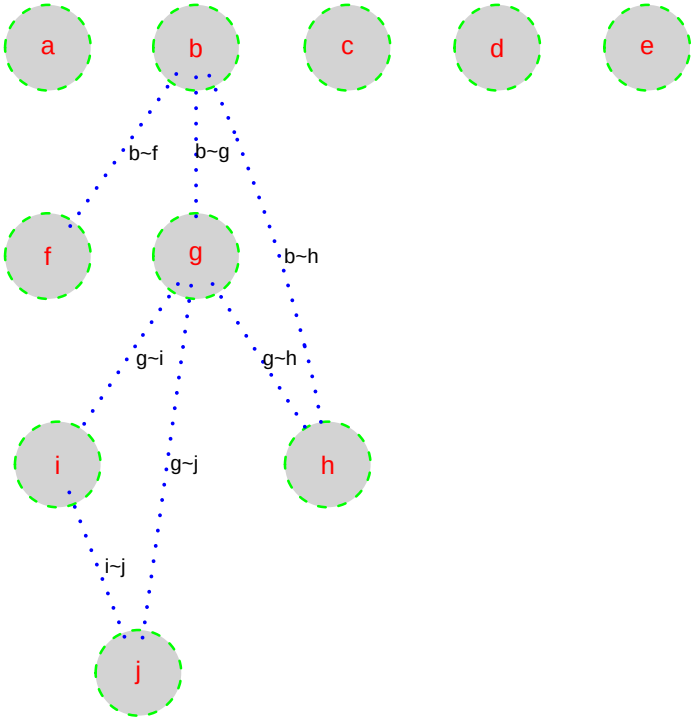
First, we set some attributes that control the edge lines.

```
> graph.par(list(edges = list(col = "blue", lty = "dotted", lwd = 3)))
> renderGraph(g1)
```



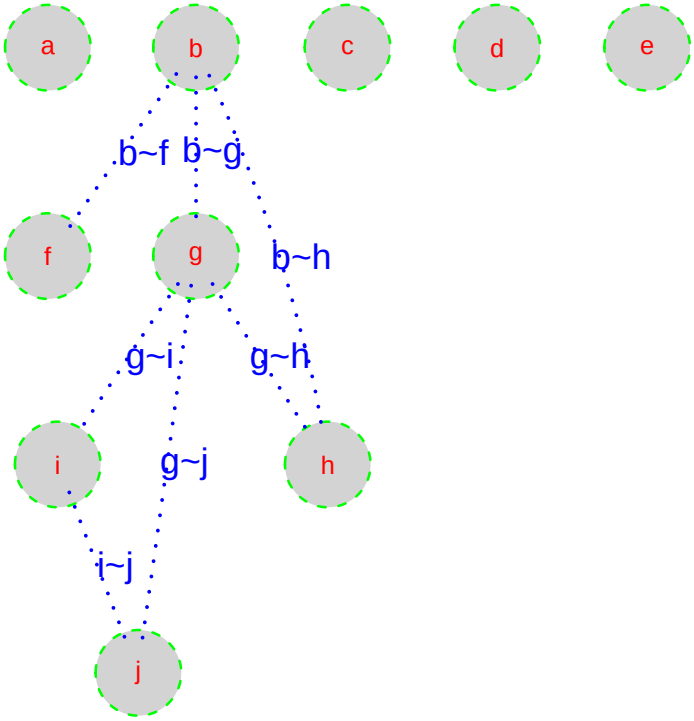
In order to show the effects of the edge label parameters, we first have to add such labels. `layoutGraph` will pass them on to *Graphviz* when they are specified as `edgeAttrs`:

```
> labels <- edgeNames(g1)
> names(labels) <- labels
> g1 <- layoutGraph(g1, edgeAttrs = list(label = labels))
> renderGraph(g1)
```



Now we can start tweaking them:

```
> graph.par(list(edges = list(fontsize = 24, textCol = "blue")))
> renderGraph(g1)
```



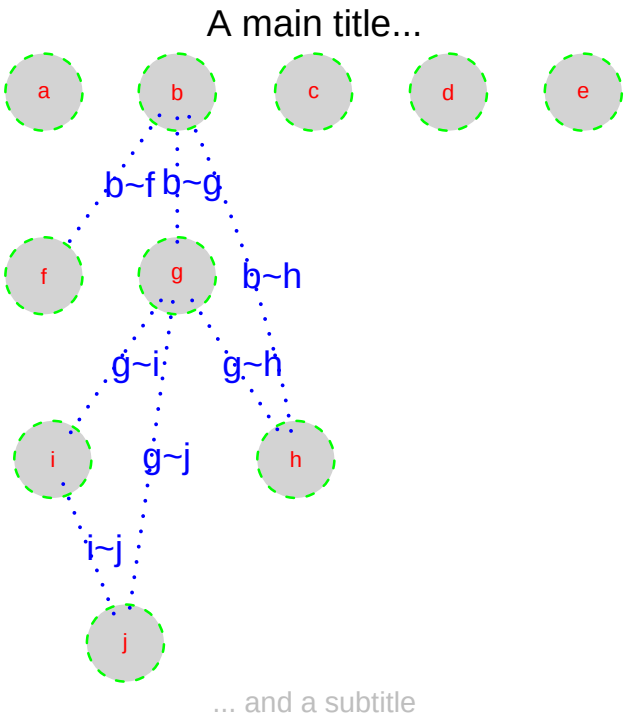
3.3 Default graphwide parameters

Some features of a graph are not really attributes of either nodes or edges. They can be controlled through the graphwide rendering parameters:

- main: text that is plotted as the main title. Unless set explicitely, no title will be plotted.
- sub: text that is plotted as subtitle at the bottom of the graph. Unless set explicitely, no subtitle will be plotted.
- lwd: the width of the edge line. Defaults to `1`.
- textCol: the font color used for the edge labels. Defaults to `black`.
- col.main: the font color used for the title. Defaults to `black`.
- cex.main: Expansion factor for the fontsize used for the title. Defaults to `1.2`
- col.sub: the font color used for the subtitle. Defaults to `black`.
- cex.sub: Expansion factor for the fontsize used for the subtitle. Defaults to `1`

Here, we add both a title and a subtitle to the plot.

```
> graph.par(list(graph = list(main = "A main title...", sub = "... and a subtitle",
+     cex.main = 1.8, cex.sub = 1.4, col.sub = "gray")))
> renderGraph(g1)
```



Of course we could set all graph-, node-, and edgewide parameters in one single call to `graph.par`. Instead of defining global settings with `graph.par` we could also provide a list with the same structure to `renderGraph` through its `graph.pars` argument. Those will only be applied in the respective rendering operation, whereas options set using the function `graph.par` are retained throughout the whole *R* session.

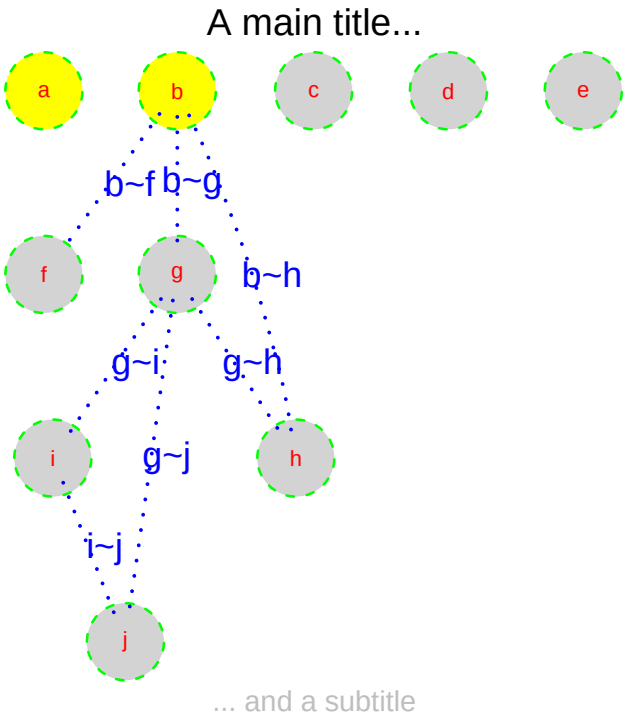
4 Parameters for individual nodes/edges

In many cases we don't want to globally change certain parameters for all nodes or edges, but rather do this selectively to highlight individual nodes/edges or subsets thereof. To this end, parameters for individual nodes and edges can be set using the `nodeRenderInfo` and `edgeRenderInfo` functions. Both `nodeRenderInfo` and `edgeRenderInfo` are replacement functions that operate directly on the *graph* object. When you change a parameter in the *graph* object this will be carried on across all further rendering and layout operations. The settings

made by `edgeRenderInfo` and `nodeRenderInfo` take precedence over all default settings.

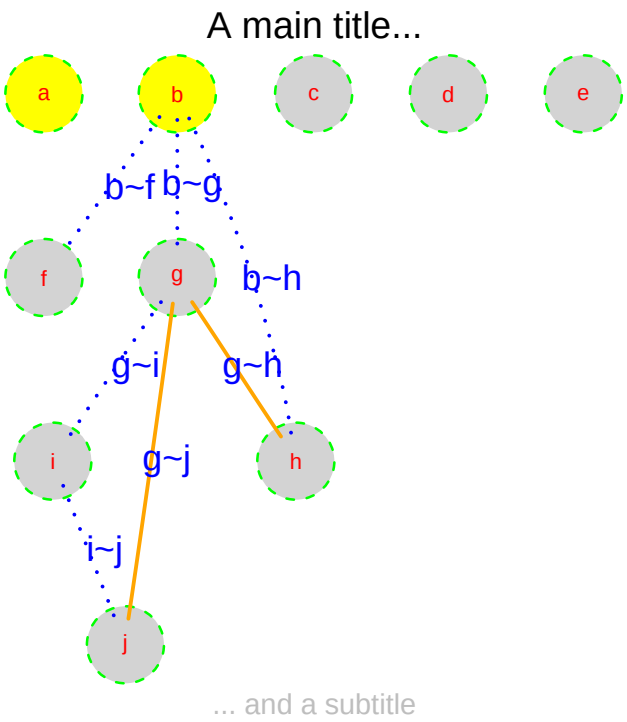
The parameters to be set have to be given as named lists, where each list item can contain named vectors for certain options. For example, the following code sets the fill color of nodes `a` and `b` to `yellow`.

```
> nodeRenderInfo(g1) <- list(fill = c(a = "yellow", b = "yellow"))
> renderGraph(g1)
```



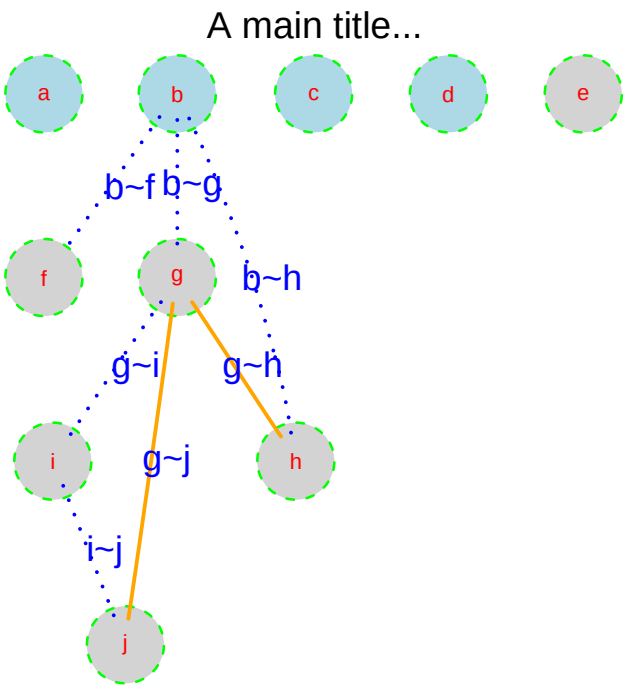
The names of the vectors have to match the node or edge names of the graph. Node names are straightforward (the result of calling the function `nodes` on a *graph* object), however edge names are made up of the names of the connected nodes separated by `~`, the tilde symbol. An edge between nodes `a` and `b` would be named `a~b`. For a directed graph `a~b` is the edge fom `a` to `b`, and `b~a` is the edge from `b` to `a`. For undirected graphs the two are equivalent. `edgeNames` returns the names of all edges in a graph. The following code changes the line type of the edges between nodes `g` and `h` and nodes `g` and `j` to `solid` and the line color to `orange`.

```
> edgeRenderInfo(g1) <- list(lty = c(`g~h` = "solid", `g~j` = "solid"),
+     col = c(`g~h` = "orange", `g~j` = "orange"))
> renderGraph(g1)
```



Changes in the rendering of specific nodes or edges is often motivated by certain classes or features they represent and we don't want to set this manually but rather use a programmatic approach:

```
> baseNodes <- letters[1:4]
> fill <- rep("lightblue", length(baseNodes))
> names(fill) <- baseNodes
> nodeRenderInfo(g1) <- list(fill = fill)
> renderGraph(g1)
```



5 Sessioninfo

This document was produced using

```
R version 2.8.1 (2008-12-22)
i386-apple-darwin8.11.1

locale:
C

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

other attached packages:
[1] Rgraphviz_1.20.4 graph_1.20.0

loaded via a namespace (and not attached):
[1] XML_2.3-0       cluster_1.11.13
```

together with version 2.12 of graphviz.