GraphLayout: graph layout as a plugin in `ggobi`

Deborah F. Swayne

May 6, 2004

Abstract

GraphLayout is a plugin for ggobi, and this manual describes its use. It can be expected to change and grow.

1   The data

First, you need some data. The ascii format supported by ggobi is not adequate for the specification of edges, so you'll either use an xml file or define the graph in R and feed it to ggobi using Rggobi.

If you use an xml file, it will contain a minimum of two datasets: a set of cases or nodes, and a set of edges. The records in the node data must have *id*s:

```
<record id="0"> ... </record>
<record id="1"> ... </record>
```

The edge records use those *id*s to specify a *source* and *destination*:

```
<record source="0" destination="1"> ... </record>
```

No two records in the same xml file may have the same record id.

The sample data that is discussed here is part of the ggobi distribution: **snetwork.xml**, an artificial social network of 140 people who are connected by 205 edges, representing some (unspecified) form of social contact. Notice that there are two variables recorded for each node, and two variables for each edge.

In a ggobi scatterplot display, use the **Edges** menu to display the edges, and maybe the "arrowheads" which indicate edge direction.

2   Specifying the datasets

Initiate the plugin by selecting the "Graph layout ..." item on ggobi's Tools menu. The GraphLayout control panel contains a "notebook" widget with three tabs. The first tab, "Specify datasets," contains two lists, one for datasets which can supply nodes and one for datasets which can supply edges. In most cases, only one choice is possible, but more complex arrangements can occur.

The *snetwork.xml* data supplied with ggobi has exactly one set of nodes and one set of edges. For this data, there are no choices to be made, and you can simply move on to select a layout method.

With the *morsecodes.xml* data, there are in fact two sets of edges. The first one, "distance," supplies the distances to be used in determining the layout. The second set, "edges," is a set of

edges that can be used for display, because it emphasizes the structure of of the data. (The layout methods in this plugin don't actually do a good job with the morse code data. It's handled much better by another plugin, called ggvis.)

3 Layout

There are three layout algorithms presently implemented in GraphLayout. A radial layout method is available without the use of any external libraries, and two more methods (called dot and neato) are available if you build GraphLayout with graphviz (www.graphviz.org).

Each layout method generates a new dataset when applied, and opens a new scatterplot displaying a plot of the node positions. The scatterplots are linked across both nodes and edges.

3.1 Radial layout

The radial layout [3] sets the first node in the dataset as the center node, and arranges the rest of the nodes in concentric circles around it. The resulting layout is a tree arranged radially, with any extra edges added. If the underlying graph is not very tree-like, the layout can result in a great many edge crossings, and the layout doesn't do anything to minimize these crossings.

Click on "apply" under the "Radial" tab. It generates a new dataset with the following variables:

- x, y: The node positions. Plot y against x to view the layout.

- depth: For each node, the number of steps required to get from it to the center node. (Plots of y or x against depth unwrap the radial layout and make it look like a tree, if an unbeautiful one.)

- in degree, out degree: The number of edges in and out of each node. (This is the only place edge direction shows up.)

- nparents: For $node_i$, the number of nodes with $depth = depth_i - 1$ with which it shares an edge.

- nchildren: For $node_i$, the number of nodes with $depth = depth_i + 1$ with which it shares an edge.

- nsiblings: For $node_i$, the number of nodes with $depth = depth_i$ with which it shares an edge.

3.1.1 Center node

To reset the center node, choose the *Identification* mode in any of the displays. (That's done most easily by typing *i* while the mouse cursor is inside the display.) While the point you want as center node is labelled, click once to make the label "sticky." The node most recently made sticky will become the center node. (Making a node "unsticky" has no effect.)

If you hide the center node, GraphLayout will refuse to proceed with a new layout until you have either restored it or chosen a new one.

3.1.2 Disjoint subgraphs

The radial layout method as implemented here can not lay out disjoint subgraphs independently. If points are encountered that do not have a path to the center node, those points will be hidden, and will not affect the layout.

3.2 Graphviz layouts

There are two layouts available with the graphviz [2] package: **dot** makes hierarchical layouts of directed graphs, and **neato** makes "spring" model layouts of undirected graphs. The software can be obtained at www.graphviz.org, and some instructions for building it for ggobi are in the INSTALL file in the GraphLayout source directory.

When you have built GraphLayout with the graphviz package, those layout methods will be available under the tabs labelled "Neato" and "Dot." With each method, a new dataset with two variables is generated (the horizontal and vertical coordinates of the node positions).

3.2.1 Neato parameters

The neato layout algorithm has some parameters.

Dimension: Neato can produce layouts in the plane (by default), and it can also produce layouts in higher-dimensional spaces, up to 10d.

Model: Neato can use one of two different models to compute the dissimilarity or distance matrix. The default is the traditional graph distance, where the number of edges in the shortest path between two nodes is the distance between them. When the "circuit resistance" model is selected, the distances are computed using a resistance circuit model. This distance reflects the normal graph distance between 2 nodes, but also considers the number of paths between them: The more paths, the smaller the distance. This metric tends to emphasize clusters more.

3.2.2 Disjoint subgraphs

Both the graphviz layout can handle disjoint subgraphs, so they do not hide points.

4 Manipulations

All the standard ggobi direct manipulations are available. Plots of the graph can obviously be linked node-wise to plots of node covariates. What might be less obvious is that they can also be linked to plots of edge covariates, so that an edge in the graph corresponds to a node in the plot of edge data. Nodes can be interactively brushed and "identified;" edges can be brushed – to set the color of the line type and width. The "color schemes" tool can be used to automatically color the nodes or the edges.

Nodes can also be moved, which can help untangle the layout a bit. Groups of nodes can be moved together: first brush them with the same symbol and color, and then select `Move brush group` in the `Move points` ViewMode.

4.1 Thinning the graph

Brushing can be used to thin the plot, by hiding nodes with especially high in or out degree, for example, or nodes with large values of depth. Once those nodes are hidden, a new layout can be produced. For the radial layout, any nodes or edges still visible that no longer have a path to the center node will then be hidden as well, so it isn't necessary to erase an entire subtree: erasing its root is enough. For the graphviz layouts, thinning the graphs may result in the creation of multiple subgraphs.

5 Plot control from R

If you have launched ggobi from within R, you can use the API to drive the plot. Here are some fragments of code in the S language that I have used.

In the first example, I have two xml files representing two related graphs, and I'm interested in comparing them. This is part of the code used to highlight the edges the two graphs have in common.

```
g1 <- ggobi("f1.xml")
setDisplayEdges.ggobi(.gobi=g1)
e1 <- getEdges.ggobi(.data=2, .gobi=g1)
g2 <- ggobi("f2.xml")
setDisplayEdges.ggobi(.gobi=g2)
e2 <- getEdges.ggobi(.data=2, .gobi=g2)
...
esame1 <- enames1 %in% enames2
edgecolors1 <- rep(1, nedges1)
edgecolors1[esame1==T] <- 6
setColors.ggobi (edgecolors1, .data=2, .gobi=g1)
...
```

In the second example, there is one graph, and its edge covariates are the values of a variable recorded for each edge at t_i. I use R to animate the graph, using color to encode the edge weight; I first chose a sequential colorscheme. Similarly, one could use setGlyphs.ggobi() to set node type or size. The same command sets edge type or thickness when applied to the edge data. To hide some of the edges, use setHiddenCases.ggobi().

```
edges <- getData.ggobi(2)
ntimesteps <- dim(edges)[2]

for (i in 1:ntimesteps) {
  tgraph (i, edges)
  ...
  colors <- integer(dim(edges)[1])
  colors[lnw>(3*mx/4)] <- 5
  colors[lnw>(mx/2) & lnw<=(3*mx/4)] <- 4
  colors[lnw>(mx/4) & lnw<=(mx/2)] <- 3
  colors[lnw>0 & lnw<=(mx/4)] <- 2
  setColors.ggobi (colors, 1:length(colors), 2)
}
```

In an extension of the second example, the xml file includes three datasets. The third is of dimension *ntimesteps* by 2, and its single time series plot represents the sum of all measurements for all edges at each time step. As the animation runs, we highlight the corresponding point in a scatterplot of this time series.

```
tcolors <- integer(ntimesteps)
tcolors[1:length(tcolors)] <- 3
tcolors[i] <- 7
setColors.ggobi (tcolors, 1:length(tcolors), 3)
```

6 Related work

As mentioned earlier, another plugin exists which can lay out graphs. It's called *ggvis*, and it uses multidimensional scaling to find the point positions. It is a port of the older *xgvis* ([1], http://www.research.att.com/areas/stat/xgobi/), which was used with xgobi, ggobi's predecessor. GGvis can perform layouts in arbitrary dimensions, and the layouts it produces can also be tuned in several ways, by adjusting the many parameters of the MDS function.

In addition, there is a plugin ("GraphAction") for manipulations that are specific to graphs.

References

[1] Andreas Buja, Deborah F. Swayne, Michael L. Littman, Nathaniel Dean, and Heike Hofmann. XGvis: Interactive data visualization with multidimensional scaling. *Journal of Computational and Graphical Statistics*, 2002 (to appear).

[2] Emden R. Gansner and Stephen C. North. An open graph visualization system and its applications to software engineering. *Software – Practice and Experience*, 30(11):1203–1233, 2000.

[3] Graham Wills. NicheWorks – interactive visualization of very large graphs. *Journal of Computational and Graphical Statistics*, 8(2):190–212, 1999.