

# ggvis: multidimensional scaling as a plugin in ggobi

Deborah F. Swayne and Andreas Buja

December 3, 2002

## Abstract

ggvis is a plugin for ggobi, and this short manual begins to describe its use. It is a port of xgvis, previously available as part of the xgobi software.

## 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 probably use an xml file with a minimum of two datasets: a set of cases or nodes, and a set of edges. The records in the node data must have *ids*:

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

The edge records use those *ids* 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 “ar-rowheads” which indicate edge direction.

## 2 Specifying the datasets

Initiate the plugin by selecting the “ggvis (MDS) ...” item on ggobi’s Tools menu. The ggvis control panel starts with 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 run multidimensional scaling.

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 point positions. The second set, “edges,” is a set of edges that can be used for display, because it emphasizes the structure of the data.

### 3 Running MDS

This section documentation has not yet been written, but we can refer you to documentation for the older *xgvis* ([?], <http://www.research.att.com/areas/stat/xgobi/>), which was used with *xgobi*, *ggobi*’s predecessor.

### 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.

### 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

Another plugin, `GraphLayout`, offers several methods for laying out graphs. One of its methods, called `neato`, produces layouts that are similar to those of `ggvis`.

## 7 Future work

A third plugin, not yet named, will allow manipulations of graphs, independent of the layout method used.