Introduction to ggvis: multidimensional scaling as a plugin in `ggobi`

Deborah F. Swayne and Andreas Buja

May 6, 2004

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 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 have at least 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 Initiating ggvis

Initiate the plugin by selecting the "ggvis (MDS) ..." item on ggobi's Tools menu. The ggvis control panel starts with a "notebook" widget with four tabs. In most cases, you will be able to jump straight to the last tab, *Run*, and start ggvis, but we will discuss the settings in each tab with reference to two sample datasets.

The *snetwork.xml* data supplied with ggobi portrays a simple social network, and we can use ggvis to lay out its graph. The *morsecodes.xml* data is the result of an experiment about rates of confusion of Morse codes which are interpreted as dissimilarities among pairs of symbols, and we can use ggvis to explore these dissimilarities. For both of these sets of data, the defaults that ggvis infers and displays in the first three notebook tabs are correct.

2.1 Specifying the datasets

The first tab, "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 social network data 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.

The Morse code data has 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 of the data.

Once the nodes and edges have been specified, move to the next tab.

2.2 Task

Under the next tab are some parameters which define the task you're performing, and perhaps set some options. First, the program wants to know whether you're going to perform *Dissimilarity analysis*, as you would with the Morse code data, or *Graph layout*, as you would with the social network data. ggvis has attempted, in a primitive way, to guess which task you're doing: If you have included an edge set named "dist," "distance," or "dissim," then ggvis guesses that you are interested in dissimilarity analysis; otherwise, it guesses that you're going to lay out a graph. You may override that guess here.

If you are in fact planning to lay out a graph, then you have two more choices to make. In the simplest graph layout situation, as in the case of the social network example, the pairwise distances are simply defined as the minimum number of edges traversed in traveling between any pair of nodes. (Edge direction is ignored.) By supplying edges, you have specified the pairwise distances for only those nodes that are directly connected; select *Complete graph distances* so that the remaining pairwise distances will be calculated.

Your second choice is whether to supply a vector of edge weights, which you can do by supplying a weighting variable in the edge dataset. The social network example includes no edge weights.

2.3 Dist

The next tab is where you will specify the source of pairwise distances (for dissimilarity analysis) or edge weights (for weighted graph layout).

For the *morsecodes* data, the target distance matrix is a confusion matrix representing the fraction of times that one character in the Morse code was confused with another in an experiment. For ggvis, we have represented that as an edge variable called **D**, which you will see has been selected by default.

2.4 Run

Once the target distance matrix D has been populated, a new scatterplot display appears. Proceed to the next tab, "Run," to begin running the multidimensional scaling algorithm.

Before clicking on the "Run MDS" button, choose the dimension of the space into which you want to project the data. Both the social network graph and the Morse code data project well into 3-dimensional spaces, but it's also interesting to see how they look in the plane.

You can now click on the "Run MDS" button to start the algorithm. You may want to start with a large step size, and then reduce the step size as the layout converges.

3 Controlling MDS

The sliders below the histogram of the target distances are quite useful in tuning the layout. Increasing the exponent of **D** increases the influences of the large distances. For the social network data, this slider controls the length of the edges close to the center and the spread of the leaf nodes.

This rest of the documentation for this section has not yet been written, but we can refer you to documentation for the older *xgvis* ([1], http://www.research.att.com/areas/stat/xgobi/), which was used with xgobi, ggobi's predecessor.

4 Comparing sets of distances or edge weights

If your input data contains two vectors of distances of edge weights, then you can pause MDS, back up to the *Dist* tab, select an alternative variable, and then begin to run MDS once more. The points in the plot in the MDS window will smoothly migrate to their new positions.

5 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.

5.1 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.

6 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)
```

7 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.

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).