DSC 2003 Working Papers
(Draft Versions)
http://www.ci.tuwien.ac.at/Conferences/DSC-2003/

Exploratory Visual Analysis of Graphs in GGobi

Deborah F. Swayne [*] **Andreas Buja** [†]

Duncan Temple Lang [‡]

Abstract

Graphs have long been of interest in telecommunications and social network analysis, and they are now receiving increasing attention from statisticians working in other areas, particularly in biostatistics. Most of the visualization software available for working with graphs has come from outside statistics and has not included the kind of interaction that statisticians have come to expect. At the same time, most of the exploratory visualization software available to statisticians has made no provision for the special structure of graphs.

Graphics software for the exploratory visual analysis of graph data should include the following: graph layout methods; a variety of displays and methods for exploring variables on both nodes and edges, including methods that allow these covariate displays to be linked to the network view; methods for thinning or otherwise trimming a large graph. In addition, the power of the visualization software is greater if it can be smoothly linked to an extensible and interactive statistics environment.

In this paper, we will describe how these goals have been addressed in GGobi through its data format, architecture, graphical user interface design, and its relationship to the R software (Ihaka and Gentleman, 1996).

[*]AT&T Labs – Research
[†]The Wharton School, University of Pennsylvania
[‡]Lucent Bell Laboratories

1 Introduction

A graph consists of nodes and edges; the edges connect pairs of nodes. In social network analysis, the nodes frequently represent people or institutions; the edges represent interactions such as conversations or trading relationships. The graphs encountered in telecommunications are similar: the nodes often represent telephone numbers or IP (Internet Protocol) addresses; the edges capture telephone calls or exchanges of packets.

For a data analyst studying graph data, the description of the graph is often only part of the story, because the nodes and the edges may each correspond to multivariate data. For example, if the graph captures a set of telephone numbers and telephone calls, we may have demographic data or usage data about the bill-payer for each telephone number, and we may also know the time and duration of phone calls. We therefore observe variables on nodes as well as variables on edges.

How do exploratory data analysts approach such data? First, we need to visualize the graph, that is, to lay it out by using node positions that have been calculated to help us interpret the graph structure. This is not a well-defined objective, but often the distance between nodes in the layout should reflect their distance from one another according to some distance metric on the graph. Another guideline is that minimizing edge crossings usually makes a graph more readable by cutting down on clutter. Still, there is no "best" layout method, or even a best layout for a particular graph: for example, one layout may clarify a graph's overall structure while collapsing local structure, while in another layout, a local region of interest may be clearly drawn but the overall structure looks like spaghetti. Graph layout in an interactive context, then, should offer several layout algorithms and a lot of interaction methods for tuning and exploration.

The layout algorithms should be fast so that they can comfortably used in real time. For example, we might draw only straight-line edges, and we might not sacrifice any time to choose the perfect position for node labels. The suite of layout algorithms should include methods for laying out graphs in 3D (or higher-D), which we can rotate to shift our viewpoint and focus on local structure.

Other important interaction methods include the following:

- We should be able to tune the layout by moving nodes interactively.
- We should be able to pan and zoom the display of the graph.
- We should have a variety of ways to thin or subset the graph by eliminating or collapsing nodes and edges. At times, we may not want to eliminate nodes, but to find ways to highlight nodes and edges of interest while "downlighting" the rest. In that way, we retain context as we focus on a subset of interest.

So far, we have considered only the structure of the graph, ignoring the multivariate data associated with the nodes and edges. Once the layout is displayed, one wants to explore the data together with the graph, to investigate the relationships between the variables and the shape of the graph. The use of linked views, by now a standard feature of interactive visualization software, is well suited to this goal.

The graph view can be linked to displays of multivariate data on both nodes and edges.

These additional views can be used to highlight, label or paint nodes and edges in the graph view according to variable values, so that we can explore the distribution of data values in the graph (see Fig. 2. Furthermore, we often want to use them to thin the graph. In the case of telephone calls, we might want to erase the edges corresponding to the shortest calls, and then erase all the nodes that no longer have edges.

Finally, this software will be more powerful and more extensible if it can be programmed using some scripting language, and if it is connected to a software system for data analysis that includes a library of standard graph algorithms.

Graph drawing is an active research area in computer science with a long history (Battista et al., 1994). The layouts produced are highly tuned and often beautiful. Since they are not produced within the context of data analysis, the graphics are typically not interactive, and the programmers have not adopted the linked views approach. Some tools (e.g. Pajek (Batagelj and Mrvar, 1998)) offer a library of graph algorithms in addition to layout, and some can even be extended with plugins (e.g. Tulip, www.tulip-software.org). Still, the designers clearly do not have exploratory data analysis (EDA) in mind.

Within the field of statistics, graph visualization has not gotten very much attention. A notable exception is the work of Wills (1999), which has never been released to the public. Even the social network analysis community, which combines an interest in graph drawing with an interest in multivariate data analysis, has not to our knowledge produced tools which combine both sets of visualization capabilities. We therefore feel that there exists a gap in current software offerings for the exploration of graph data. GGobi is our attempt at to fill this gap.

This paper is structured as follows. Section 2 introduces GGobi, the software which will be discussed in the rest of the paper. Section 3 describes GGobi's methods for graph layout. Section 4 describes some of GGobi's methods for manipulating displays, especially graph views. Section 5 explains how GGobi can be embedded in other software, and what this design offers for graph data analysis. Section 6 describes the data format that is used to specify relationships between nodes and edges, graph elements and variables. We use a real telecommunications dataset for illustration throughout the paper. The meaning of its variables has been masked to protect the privacy of the customers.

2 GGobi

GGobi (Swayne et al. (2003)) is general-purpose multivariate data visualization software, designed to support EDA. GGobi displays include scatterplots, scatterplot matrices, barcharts, time series plots, and parallel coordinate plot. All displays can be linked for color and glyph brushing as well as for point and edge labeling. GGobi is known for its powerful projection facilities for high-dimensional rotations. Among GGobi's many other manipulations are panning and zooming, subsampling, and interactive moving of points and groups of points in data space.

GGobi can be easily extended, either by being embedded in other software or by the addition of plugins; either way, it can be controlled using an Application Programming Interface (API). An illustration of its extensibility is that it can be embedded in R.

GGobi is a direct descendent of a data visualization system called XGobi (Swayne et al., 1998) that has been in use since the early 1990's. XGobi supported the specification and display of graphs, but it did not include any graph layout methods. Graph data was an afterthought with XGobi, while it was a consideration in the GGobi design process from the beginning.

GGobi supports a plain ASCII format involving multiple input files (as in XGobi) for the simplest data specifications, but an XML (Extensible Markup Language) file format has to be used for anything richer, and graphs are an example. The format is briefly described in Section 6.

3 Graph layout

We have used GGobi's plugin mechanism to add graph layout. Because this is specialized software, it is convenient that this functionality can be optional. There are two plugins available for GGobi that can be used for laying out graphs.

3.1 The graph layout plugin

The simplest plugin is called GraphLayout. It includes three layout methods, two of which rely on the library included with GraphViz (Gansner and North, 2000), a freely available collection of tools for manipulating graph structures and generating graph layouts. All three methods work by generating a new dataset on the fly and making it available through the GGobi interface, so scatterplots of the new position variables can be displayed, and edges added to them.

The three layout methods are:

Radial: The radial layout (Wills, 1999) places a designated node at the center, 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. In addition to the two position variables, the method generates a few other variables, such as the number of steps between node j and the center.

Dot: "Dot" produces hierarchical layouts of directed graphs in 2D; the other layout methods ignore edge direction. It first finds an optimal rank assignment for each node, then sets the vertex order within ranks, and finally finds optimal coordinates for the nodes.

Neato: The "neato" layout algorithm produces "spring" model layouts of undirected graphs. In spring models, the graph is modelled as a set of objects connected by springs, assuming both attractive and repulsive forces, and an iterative solver is used to find a low-energy configuration. Only the positions at the final configuration are returned by the algorithm. Neato is the most general-purpose method of

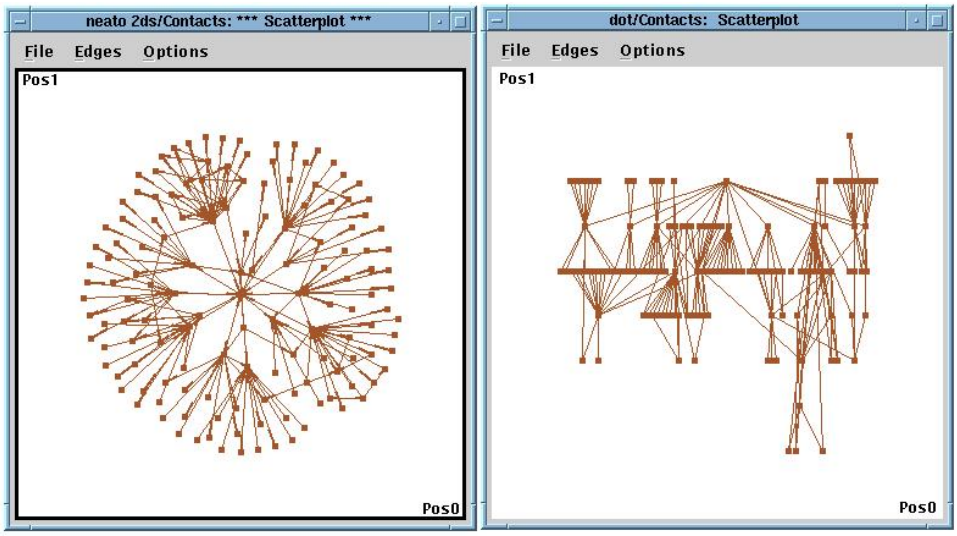

Figure 1: These two displays show layouts of the *snetwork.xml* data generated by the GraphViz layout methods. On the left is a 2-D "neato" layout; on the right a "dot" layout.

the three. Further, neato can generate layouts in spaces from 2D to 10D, and edge weights can be used to further tune the layout.

The first layout method is illustrated in Fig. 2; the latter two are illustrated in Fig. 1.

There is a manual for the plugin which describes its use in more detail. The dot and neato layout methods are described in the GraphViz documentation, which can be found on `www.research.att.com/sw/tools/graphviz/refs.html`.

3.2 The ggvis plugin: multidimensional scaling

The "ggvis" plugin is a reimplementation of XGVis (Buja and Swayne, 2002), a multidimensional scaling (MDS) tool which is part of the XGobi software. MDS is a method for visualizing data where objects are characterized by dissimilarity values for all pairs of objects. It interprets these dissimilarities as distances and constructs maps in R^k. It was originally developed as a data analysis method in the social sciences, but it is also used to lay out graphs.

Like neato, ggvis computes layouts through iterative optimization, but unlike neato, the display is redrawn at each iteration, so we can watch the layout take shape. We can also intervene during the optimization process, by moving points interactively when they are trapped in local minima, or by adjusting parameters of the MDS objective function.

GGVis puts a large number of parameters under interactive user control. As a consequence, ggvis layouts are highly tunable. One of the most useful ggvis param-

eters is the exponent of a power transformation of the target distances; lowering it below one lets the short distances dominate, while exponents greater than one expose the long distances. This lets us decide whether we want to spread the leaves out, highlighting the structure in the leaves, or to collapse them, revealing the connectivity in the interior of the graph.

In addition to parameters, we can make use of color and glyph groupings of the nodes. We may subselect one group at a time for layout, or we may lay out the groups simultaneously but as unconnected graphs. Or we may lay out a subgroup and use it as an achor set for laying out the remaining nodes.

There is also a diagnostics plot that permits us to judge the quality of the layout in terms of matching the target distances.

3.3 Multiple edge sets

Sometimes one wants to compare different edge sets for the same set of nodes. In the case of telephone calls, for instance, the extended community associated with a phone number changes from week to week, with changes both in the set of phone numbers in the community, and in the total length of the conversations between any pair of nodes.

One strategy to compare these different edge sets is to start by determining a layout based on the union of all nodes and the union all of edges. Since any of the edge sets can be associated with the set of nodes used to determine the layout, it's easy to compare them: Open multiple scatterplots of the nodes in the graph view, and assign a different edge set to each one. That technique could even be the basis for an animation of edges and edge variables over time.

4 Graph exploration

Once the layout has been produced and the graph is displayed, a great deal of exploration is possible without using any further plugins. Most of this functionality depends on using linked views. As one would expect, nodes in the graph view are linked to points in scatterplots of node variables, or to bars in a barchart; this is a familiar style of linking. It is perhaps less obvious that an edge in the graph view and a point in a scatterplot of edge variables are also linked: these are just different ways of rendering the same record. Here are some of the manipulations available in GGobi:

Move Points: In this mode, any point can be moved to manually tune the layout. To move a group of points, one brushes them with a common glyph and color; by moving any member of the group, one moves the whole group. Under certain circumstances, point motion can be linked across plots of layouts, namely, when the nodes are shared across graphs that differ only in edge sets and share a single layout in separate windows.

Edit Edges: To edit the graph interactively, add nodes (by clicking the mouse where you want the new node to appear) and edges (by pressing down the mouse button at the source node and dragging the edge to the destination). To view or

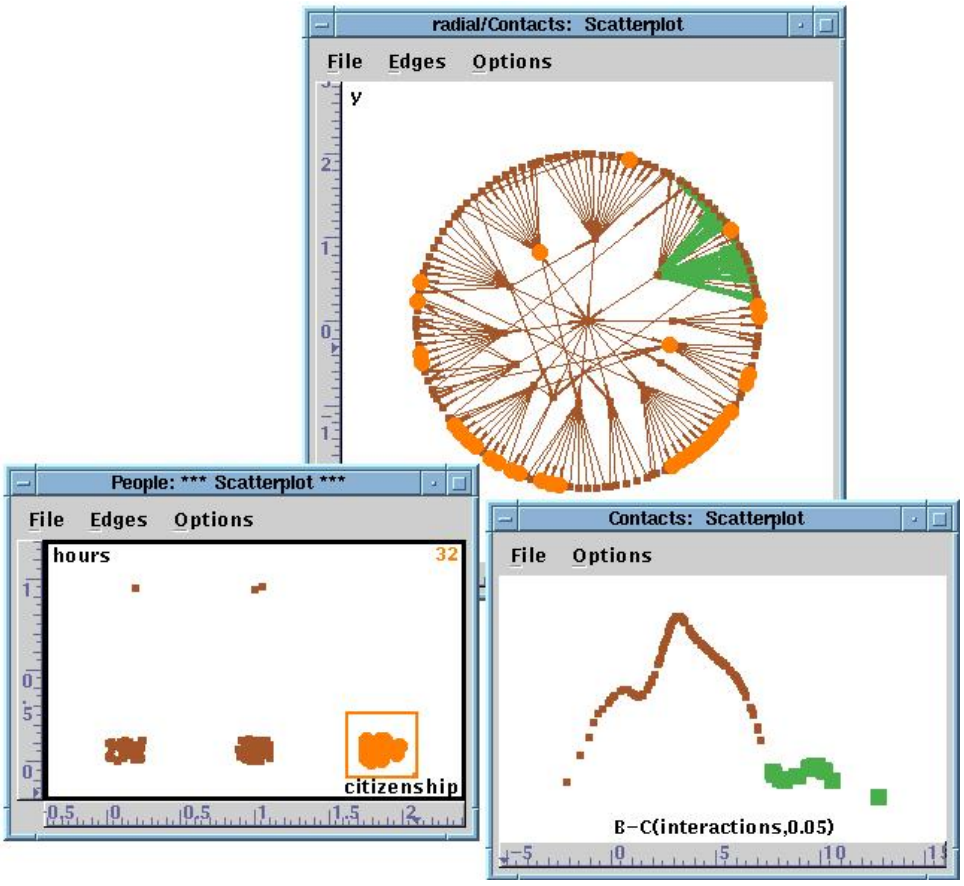

Figure 2: An illustration of linked brushing with graphs. The nodes in the graph are linked to the data in the scatterplot at the lower left; the edges to the data in the scatterplot at the lower right.

modify the default properties (such as record label or variable values), use the left button; to simply have the new record added quickly, use the right or middle button. To delete nodes or edges, use "shadow" brushing as described below.

Identify: When the identification mode is active, bringing the cursor near a point causes a label to be displayed, both in the current display and in other displays. By default, this is the case label supplied in the data file (or the row number), but it can also be a list of variable name - value pairs or an id. If edge identification is selected, the nearest edge will be labelled instead of the nearest point.

Brushing (interactively): Linked brushing is probably the most familiar use of linked views. In the case of graphs, it is probably clear by now that it can be used in at least two ways. First, a plot of node data is linked to a graph view such that brushing points in one plot causes the same points to change color or glyph

in the other. Second, a plot of edge data is linked to the graph view such that brushing points in the edge data plot affects the edges in the graph view, and vice versa. This latter functionality is an innovative feature of ggobi.

One brushing style allows a point or an edge to be "shadow" brushed, so that it's drawn in a faint color and can be removed from the displays altogether.

Fig. 2 shows linking between a radial layout of the *snetwork.xml* data and two scatterplots. Two rectangular arrays of data are involved, one for the nodes and the other for the edges. The window at the lower right contains a 1-D plot (an ASH, or Average Shifted Histogram) of a transformation of one of the edge variables, *interactions*. The highest values have been brushed with large green rectangles, and the corresponding edges in the radial layout view are wide and green. The window at the lower left contains a jittered scatterplot of *hours vs citizenship*, the two variables recorded for each person. The points representing the people with the highest values of the citizenship variable are being brushed with large orange circles, and corresponding points are brushed in the graph view.

The line characteristics (color, type and thickness) are implied when the point characteristics (color, type and size) are specified in the *Choose color & glyph* panel.

One of the options available in the brushing mode is shadow brushing (Becker and Cleveland, 1987); that is, to select points or edges to be drawn in a "shadow" color, close to the color of the background. This is especially appealing for graph visualization because clutter is often severe, yet we often don't want to lose sight of the graph structure when viewing a subset of the data. (Sometimes, of course, we don't want to draw those points at all, even as shadows, and then we exclude them using the *Color & glyph groups* tool.)

Coloring by variables: Since interactive brushing of continuous variables can be tedious, an automatic scheme is available as part of the *Color schemes* tool. In the *snetwork.xml* data, one of the edge variables (*interactions*) is continuous, so we choose a sequential color scale and apply it to the "Contacts" edge set using the "interactions" variable. (Since the distribution of that variable is highly skewed, we might also apply a transformation first.)

Panning and Zooming: It is essential to be able to zoom in on interesting regions of the graph view, and that functionality is available in GGobi's scale mode. (GGobi displays are not linked for scaling.)

All these methods are described in more detail in the GGobi manual, available on `www.ggobi.org`.

4.1 The graph manipulation plugin

All of the interactive methods just listed are useful for multivariate data, not just for graphs. In addition to those methods, we have added a plugin for methods of exploration that are peculiar to graphs. It has two functions as of this writing, both of them designed for focussing on contiguous subsets of the graph.

The first function hides leaf nodes recursively until a sort of core graph remains. It can be a useful way to quickly hide a lot of clutter in a a messy graph.

The second is a method for focussing on a node and its nearest neighbors. It is used in conjunction with the *Identification* mode in GGobi. Move the cursor near a

point of interest, and then click a mouse button. All points will be shadow brushed with the exception of the nearest point and its neighbors within one or two steps. In this way, one can walk around the graph, focussing on one small neighborhood at a time.

5 Graphs in GGobi's API

While GGobi is a stand–alone application, it has been designed and constructed as a programming library and can be embedded within other applications. It has a large, and still evolving, Application Programming Interface (API) which developers can use to integrate the GGobi functionality with other code. For data analysts, GGobi becomes much more powerful once it is embedded in a statistics environment with an extension language.

Our most developed example is the Rggobi package, which allows GGobi to be embedded in the R process. Users can then launch GGobi (using R data frames or data files outside R), and then get and set data values, the state of various attributes (such as color and glyph), and even add event handlers which cause R to respond to GGobi events. Edge sets can also be added, and the attributes of edges (color, line type and line thickness) are handled exactly like point attributes.

In this first simple example, we create a matrix to represent the nodes, and open it in ggobi. We next create an empty data set, dimensioned to hold six records. Finally we create a $6x2$ array to define the edges as 6 rows of source - destination pairs, named in terms of the node labels, and add the edge set to the running ggobi.

```
x <- matrix(c(0,0,2,1, 0,2,0,1, 0,0,0,1), 4, 3,
     dimnames = list(c("a", "b", "c", "d"), c("X", "Y", "Z")))
gg <- ggobi(x)

d2 <- gg$createEdgeData(6, name="edges")
e2 <- rbind(c("a","b"), c("b","c"), c("a","c"),
            c("a","d"), c("b","d"), c("c","d"))
gg$setEdges(e2, edgeset = d2)
```

In the second example, we deal with a more complex case, in which there are variables corresponding to the edges as well as to the nodes. We start again, using the matrix x just described. Next we add a second dataset, $3x2$, composed of the data corresponding to the edges. Finally we add 3 edges to the second dataset.

```
gg <- ggobi(x)

z <- matrix(c(1,2,1, 1,2,2), 3, 2,
     dimnames = list(letters[10:12], c("X", "Y")))
d2 <- gg$setData(z, name="z")

e1 <- rbind(c("a", "b"), c("b", "c"), c("a", "d"))
gg$setEdges(e1, edgeset=gg[["z"]])
```

We plan to extend the API and the Rggobi package so that they can work with other graph packages currently under development as part of the Bioconductor project (`www.bioconductor.org`).

6 Data format: Specifying graphs in XML

GGobi relies on XML (the Extensible Markup Language) for everything beyond the simplest of input data. The use of XML has allowed us to design a system of mark-ups or tags that describe one or more datasets in great detail within a single file, even specifying the relationships between records in different datasets.

We based GGobi's XML format on a pre-existing XML format designed for the Omegahat project (`www.omegahat.org`) and the S language (R and S-Plus). Some of the information that can be specified in the GGobi XML file includes variable types and axis ranges, the symbol and color corresponding to a record, and multiple data sets and the rules for linking them.

GGobi's XML format is described elsewhere (Temple Lang and Swayne, 2001), so we will only explain here how the specification of data records is used to describe graphs. A data record specification may be as simple as this:

```
<record> 1.0 2.5 </record>
<record> 1.7 2.2 </record>
```

This is a pair of records for a dataset with two variables. If we want to identify these records as nodes, we must also give them unique ids. (Ids can also be used for linking and identification, but that usage is described elsewhere.)

```
<record id="Macbeth"> 1.0 2.5 </record>
<record id="Banquo">  1.7 2.2 </record>
```

If we want a set of edges to be drawn on a scatterplot or a graph view of these nodes, we need a second dataset. If there is to be an edge from "Macbeth" to "Banquo," the second dataset must contain a record like this:

```
<record source="Macbeth" destination="Banquo"> </record>
```

If there are variables corresponding to that edge, they are specified within the record, just as they are for nodes.

```
<record source="Macbeth" destination="Banquo"> 27 42 4.6 </record>
```

As we implied in Section 3.3, it's possible to specify more than one edge set corresponding to the same node set within the same XML file, and that offers a way to compare related edge sets.

There are graph specification languages in XML under development, and we expect it will be easy to translate between those formats and GGobi's, though those other languages probably won't support multivariate data.

For the interested reader, the GGobi distribution includes several graph datasets in XML. Some include position variables so that additional layout isn't required:

buckyball.xml and *cube6.xml* describe geometric objects, with no additional variables. Another, *snetwork.xml,* is fully multivariate and does not include variables that can be used for displaying the graph; that is the dataset that served as an example throughout this paper.

7 Conclusions

As more statisticians become interested in graph data analysis, they approach this area with the expectations and expertise acquired in working with general multivariate data. They expect first of all to be able to work in environments like R, with a set of algorithms, a variety of static display methods, and a scripting language. This set of goals is being pursued in the Bioconductor project and elsewhere.

Second, statisticians and other data anlysts who have come to rely on direct manipulation graphical methods will want to use them with this form of data as well: to quickly update plots, changing variables and projection, to pan and zoom displays, and to use linked views to explore the graph and the distribution of multivariate data in the graph. GGobi's data format supports describing the graph and the data together, and its architecture allows the addition of plugins, so it's natural to extend GGobi, applying all its functionality to graph data.

Finally, we want to integrate the direct manipulation graphics, algorithms and scripting language so that we can use them all together. This expectation is not yet as automatic as the first two: People often still imagine building a single monolithic application that can do everything. As the example of graph data shows, however, there are many specialized problems that are often overlooked, so no monolithic piece of software can satisfy the needs of all users. If instead it's possible to integrate complementary software tools, and to extend them with plugins and packages, then even the most unusual cases can be handled without too much trouble.

The GGobi software and documentation, including several plugins and the Rggobi package, are available on the web site `www.ggobi.org`.

References

Batagelj, V., Mrvar, A., 1998. Pajek - program for large network analysis. Connections 21, 47–57.

Battista, G. D., Eades, P., Tamassia, R., Tollis, I., 1994. Annotated bibliography on graph drawing algorithms. Computational Geometry: Theory and Applications 4, 235–282.

Becker, R. A., Cleveland, W. S., 1987. Brushing scatterplots. Technometrics 29, 127–142.

Gansner, E. R., North, S. C., 2000. An open graph visualization system and its applications to software engineering. Software – Practice and Experience 30 (11), 1203–1233.

Ihaka, R., Gentleman, R., 1996. R: A language for data analysis and graphics. Journal of Computational and Graphical Statistics 5, 299–314.

Swayne, D. F., Cook, D., Buja, A., 1998. XGobi: Interactive dynamic data visualization in the X Window System. Journal of Computational and Graphical Statistics 7 (1), 113–130.

Swayne, D. F., Temple Lang, D., Buja, A., Cook, D., 2003. Ggobi: Evolving from xgobi into an extensible framework for interacive data visualization. Journal of Computational Statistics and Data Analysis To appear.

Temple Lang, D., Swayne, D. F., 2001. The ggobi XML input format. Www.ggobi.org.

Wills, G., 1999. NicheWorks – interactive visualization of very large graphs. Journal of Computational and Graphical Statistics 8 (2), 190–212.