GGobi: Evolving from XGobi into an Extensible Framework for Interactive Data Visualization

Deborah F. Swayne [a] Duncan Temple Lang [b] Andreas Buja [c]
Dianne Cook [d]

[a] *AT&T Labs - Research, Florham Park NJ*
[b] *Lucent Bell Laboratories, Murray Hill, NJ*
[c] *The Wharton School, University of Pennsylvania, Philadelphia, PA*
[d] *Iowa State University, Ames, IA*

Abstract

GGobi is a direct descendent of a data visualization system called XGobi that has been around since the early 1990's. GGobi's new features include multiple plotting windows, a color lookup table manager, and an XML (Extensible Markup Language) file format for data. Perhaps the biggest advance is that 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 API (Application Programming Interface). An illustration of its extensibility is that it can be embedded in R. The result is a full marriage between GGobi's direct manipulation graphical environment and R's familiar extensible environment for statistical data analysis.

Key words: Statistical graphics, interoperability, R, XML, API, plugins

1 Introduction

This paper describes GGobi, an interactive and dynamic software system for data visualization. GGobi is the result of a significant redesign of the older XGobi (Swayne et al., 1992, 1998) system, whose development spanned roughly the past decade. GGobi differs from XGobi in many ways, and it is those differences that explain best why we are undertaking a radical redesign. Therefore this paper is largely structured as a comparison of GGobi and XGobi. Here are some of the main points:

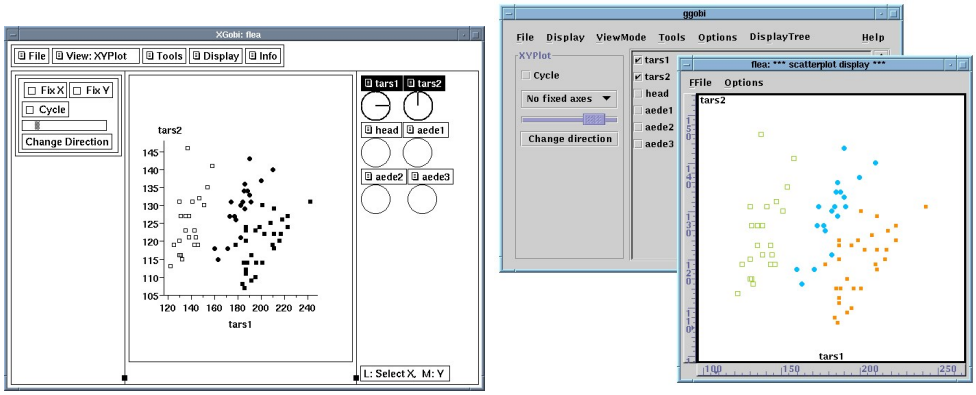

Fig. 1. A comparison of XGobi (on the left) and GGobi. The most obvious differences are that GGobi now uses a multi-window design and a new toolkit. The basic elements are still similar.

- GGobi's appearance: GGobi's appearance has changed in several ways, some of which can be seen in Figure 1: 1) It uses a different graphical toolkit with a more contemporary look and feel and a larger set of components. The new toolkit is called `GTK+`, which explains the initial G in GGobi. 2) With XGobi, there is in general a single visible plot per process; to look at multiple views of the same data, one launches multiple processes. A single GGobi session can support multiple plots, and a single process can support multiple independent GGobi sessions. 3) While XGobi supported essentially a single scatterplot and a subordinate parallel coordinate plot, GGobi supports several types of plots: scatterplots, parallel coordinate plots, scatterplot matrices, and time series plots.

  Other changes in GGobi's appearance and repertoire of tools include interactive tools to specify and tune color maps, the ability to add variables on the fly, and a new interface for view scaling (panning and zooming).

- Portability: A major advantage of using the new toolkit (`GTK+`) is portability. It originated in the Linux®community, but it has been ported to Microsoft Windows™and is being ported to the Macintosh®. It was complicated to run XGobi on a machine running Windows, because it required installation of an X Window System server. GGobi, on the other hand, runs directly under Windows.

- GGobi's data format: GGobi's data format has been extended significantly from that of XGobi. To describe a set of data for the older XGobi, one creates a set of files with a common base name, with the data in one file, and other files for the variable names, case labels, point colors, point glyphs, lines, and so on. GGobi continues to support this scheme in a limited way.

Linux is a registered trademark of Linus Torvalds.
Microsoft Windows is a trademark of Microsoft, Inc.
Macintosh is a registered trademark of Apple Computer, Inc.

However, the new format supports many new features that are too complex to describe in the multiple file format.

This new format uses a single file in XML™, the Extensible Markup Language, which is emerging as a standard language for specifying structured documents and data formats. The use of a single file aids consistency of the different elements of the input, making it easier to validate and maintain. An XML document looks similar to an HTML document, but it allows one to introduce new markup elements. The availability of tools in many different programming languages to read and write XML will help in using a single data file across different applications. The use of XML in GGobi allows complex characteristics and relationships in data to be specified. For example, multiple datasets can be entered in a single XML file, and specifications can be included for linking them. Using the XML format, GGobi can read compressed files and can read files over the network.

- Embedding GGobi in other software: GGobi's ability to interface with other software has become much richer than XGobi's. The designers of XGobi had always intended that it be used in conjunction with some analytical software, but XGobi offers very limited support for such use – one can launch an XGobi process from the S command line, but no live connection is maintained. Interprocess communication was used to maintain a live connection between XGobi and ArcView, but the same approach couldn't be easily used with S. By contrast, GGobi can be treated as a *C* library and directly embedded in (or linked with) other software, then controlled using an API (Application Programming Interface). This allows GGobi functionality to be integrated into one's own stand-alone application and also provided as an add-on to existing languages and scripting environments. For example, it can be controlled from the R command line or from a Perl or Python script. One can think of this relationship as adding interactive dynamic graphics to the environment of the scripting language, or as adding a programming language and a set of libraries to GGobi.

- Extending GGobi with plugins: Another way to extend GGobi is by developing "plugins". The plugin mechanism allows the authors and other developers to provide add-on extensions to GGobi that are not part of the core design. These might introduce new plot types, new ways to read data (such as databases and files with special formats), or auxiliary tools to view and manipulate data. This facility, in combination with the GGobi API, makes it feasible to create customized versions of GGobi for different audiences and data analysis contexts.

The rest of the paper is structured as follows. Section 2 introduces XML and describes GGobi's data format. Section 3 briefly describes GGobi's graphical user interface (GUI), and then discusses in more detail some of the differences between GGobi's GUI and data management and those of XGobi. Section

4 describes how GGobi can be compiled into a library, embedded in other software and controlled using an application programming interface (API). Section 5 describes the use of plugins to extend ggobi. Finally, section 6 introduces GGobi with an example. In the first part of the example, GGobi is used alone for a preliminary investigation of some data; the second part illustrates how GGobi and R are can be used in conjunction. (R (Ihaka and Gentleman, 1996) is a freely available implementation of the S language (Becker et al., 1988).)

2 GGobi's data format

The XGobi input format uses a set of files with a common base name and different extensions to specify data, variable names, record labels, point colors and glyphs, edges, connections between records, etc. It has the advantage that each file is easy to describe and read, with line i in one file corresponding to line i in several other files. This simple scheme has its disadvantages, too. If one file is changed, all the others may have to be changed, so it is difficult to extend or modify a set of files. More importantly, it is difficult to specify any other relationships among files than this simple line-to-line correspondence.

GGobi will continue to support this scheme for basic data input: the data, variable and case labels, and point glyphs and colors ("foo.dat", "foo.col", "foo.row", "foo.glyphs", "foo.colors").

The new XML file format makes datasets easier to share and maintain and allows the specification of more complicated relationships within a single dataset and between datasets. It has the single disadvantage that it is no longer easily human-readable, but we consider that a minor concern when compared with the advantages.

2.1 What is XML?

XML, the Extensible Markup Language, is a way to annotate or mark up both content and data. An XML file looks a lot like an HTML document at first glance, but is richer as it allows the use of new tags or elements rather than providing only a fixed set of tags (e.g., B, H1). The motivation for using XML is that it allows one to include meta-information (such as a description of how the data were recorded), parameters describing the structure of the data (such as the number of records) and other auxiliary details along with the data itself. In this respect, XML is like many specialized markup languages. However, the power of XML is that it provides a common infrastructure or scheme within

which one can easily define new data formats, simply by introducing new elements and relationships between these elements. In other words, XML is a meta-language since it allows one to define new markup languages. Each of these languages can be easily processed using the same high-level parsing tool, but each application can interpret the content in an application-specific manner. Additionally, an application can ignore content it doesn't understand. This makes it easy to extend a data format based on XML without interfering with existing software. XML is rapidly becoming a standard and already a large collection of tools for processing and generating XML documents are available for all of the common programming languages. This allows us to use the same XML data input within many applications and also significantly reduces the effort of creating software to read specialized file formats.

XML is becoming increasingly widespread on the Web and is also used for more traditional data storage and exchange. It is becoming popular for communicating genetic, geographic, graphical and business data, to list just a few areas in which it is used. XML also offers the ability to validate the structure of a document, element by element, without having to process it with the software with which it will be read. This greatly minimizes one source of software and data error and separates preparation of inputs from the software itself.

2.2 XML in GGobi

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 S languages (R and S-Plus). Some of the information that can be specified in the GGobi XML file includes:

- Variable types: Instead of trying to make the software guess whether a variable is categorical or continuous, we can specify its type. For this, we use the tags *realvariable* and *categoricalvariable*.
- Variable axis ranges: By providing *min* and *max* attributes within the *realvariable* element, a GGobi user can specify the limits to be used for creating variable axes, allowing related variables to be shown on the same scale. We approximated this functionality using a *.vgroups* file in XGobi.
- Multiple related datasets: Just as a database usually consists of more than one relational table, a visual data analysis project often consists of multiple datasets. It is convenient to be able to store them in a single data file and examine them within a single GGobi session.
- Linking between datasets: In the simplest relationship, a record in one

dataset represents the same subject as a record in another, and the datasets have the same number of records. In more complex situations, one dataset may contain data on only a subset of the data in another dataset, or a record in one dataset may correspond to a group of records in another. Linking in GGobi is based either on case identifiers or on the values of categorical variables. The tags for specifying them in a GGobi XML file are *id* and *categoricalvariable*. Section 3.4 has more details on linking.

- Graphs: One of our reasons for writing GGobi is our need for new tools for exploratory visualization of networks and graphs in general. An elementary requirement for graph visualization is (roughly speaking) linking points to lines. This arises as follows:

  Measurements on networks often involve two linked datasets. The first represents measurements on nodes, where a node may be an ethernet address or a person within a social network. The second represents measurements on edges, where an edge is one or more transactions between a pair of nodes: an exchange of packets, say, or a social contact. The correspondence of edges and pairs of nodes suggests linking scatterplots of node data and scatterplots of edge data such that highlighting of a point in the edge plot causes instant highlighting of the line connecting the two corresponding points in the node plot. Thus queries about transactions (i.e. edges) yield immediate answers in terms of the nodes involved in the transactions.

This example illustrates the use of XML to describe two related datasets in a single file. The example is a tiny artificial social network of four people, and Figure 2 is a snapshot of a GGobi session exploring the sample data.

```
<ggobidata count="2">
 <data name="Employees">
  <variables count="2">
   <realvariable name="level" />
   <realvariable name="salary" />
  </variables>
  <records  count="4">
    <record id="0"> 0 15000 </record>
    <record id="1"> 1 25000 </record>
    <record id="2"> 1 19000 </record>
    <record id="3"> 2 40000 </record>
  </records>
 </data>
 <data name="Contacts">
  <variables count="1">
   <realvariable name="frequency" />
  </variables>
  <records  count="3">
   <record source="0" destination="1"> 8 </record>
```

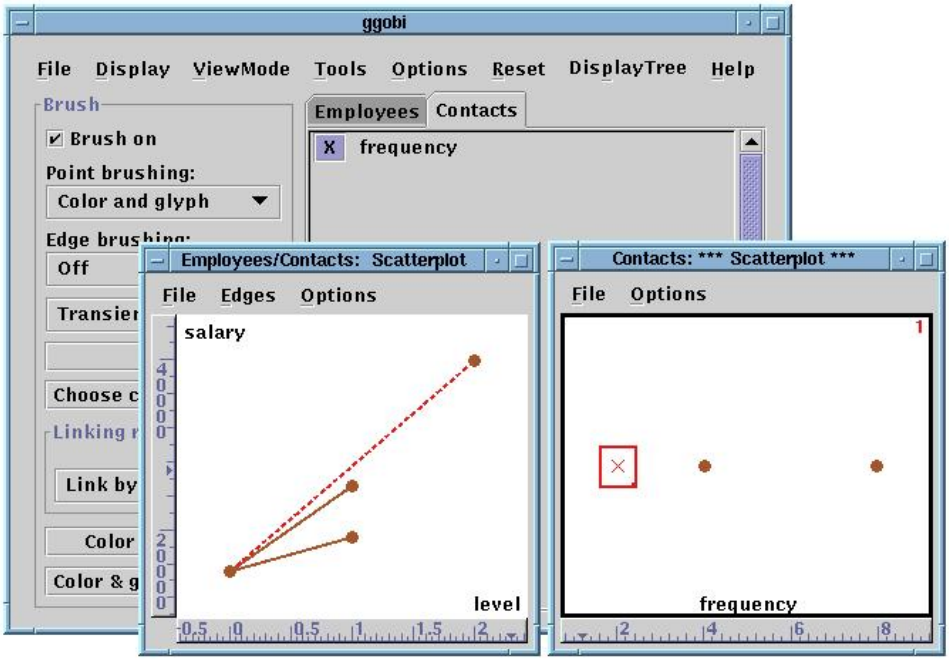

Fig. 2. A GGobi display for the XML sample data illustrating a tiny social network of four people. The GGobi console is in the background: note the tabs appearing above the variable selection area (the checkboxes), allowing separate variable selection for either dataset. The leftmost plot shows a scatterplot of the two variables in the "Employees" dataset, with edges added to illustrate which employees have contact with each other. The rightmost scatterplot shows a 1D plot of the frequency of contact. Using brushing, we see that the lowest frequency of contact occurs between the employees whose salaries and levels differ the most.

```
   <record source="0" destination="2"> 4 </record>
   <record source="0" destination="3"> 2 </record>
  </records>
 </data>
</ggobidata>
```

3 GGobi's graphical user interface

GGobi's graphical user interface uses a toolkit called `GTK+` (see www.gtk.org). It is used extensively by Linux programmers, which ensures a high level of quality and state of the art GUI technology. The toolkit has been ported to other platforms, though, making GGobi more portable than XGobi. It also provides GGobi with a cleaner and look and feel than XGobi, since XGobi was written using a toolkit whose development was frozen in the early 1990s.

In the following sections we discuss a few GUI-related topics, such as the supported display types and their operations, and the linking model for multiple displays.

3.1 Types of displays and their operations

XGobi's display types and their operations have been extended in GGobi. XGobi's two central display types, 1-D views in the form of dot plots and density plots, and 2-D views in the form of scatterplots, are both part of GGobi, but GGobi adds scatterplot matrices and time series plots. While XGobi has a limited parallel coordinate display without interactivity, GGobi has a parallel coordinate display with many of the same interactions that are available in scatterplots, such as color brushing and case identification (labeling).

XGobi's more distinguishing features, the powerful projection facilities, are available in GGobi as view modes of scatterplot displays: 2-D grand tours, 2×1-D correlation tours. New is a 1-D tour for density plots. The tours feature some novel intuitive GUI controls.

Other data and plot manipulation tools found on XGobi's "Tools" menu are available in GGobi too: variable transformation, jittering to separate ties, missing value facilities for imputation, displays for exploring missing value patterns, subsetting by eliminating color and glyph groups, and subsetting by subsampling or by the selection of blocks of cases. The actions of these tools apply to all display types equally.

3.2 Multiple displays – controls

The organization of displays has changed drastically from XGobi to GGobi. Roughly speaking, XGobi has one display per process but GGobi permits multiple displays per process, in fact as many as a user desires. When multiple displays are needed, an XGobi user starts up multiple processes, but a GGobi user opens multiple displays from within the same process. This difference in design has broad implications in two areas: display controls and linking multiple displays.

XGobi's way of controlling multiple displays is simple: every display has its own set of controls. This scheme is simple from a GUI perspective because there is never any confusion about which controls act on which displays; the scheme is expensive, though, in terms of screen real-estate.

GGobi, by comparison, controls multiple displays from a single detached control window or console. This console acts on only one data display at any given time. To make this work, two elements must be provided: 1) a way of visually indicating which data display is currently being controlled, and 2) a way of switching the data display that is being controlled. These issues are

addressed as follows: a user clicks on a data display and thereby makes it the one being controlled; at the same time there appears a thick framing rectangle around the border of the selected plot. Part of GGobi training is to develop an awareness of the visual cue that singles out the currently controlled data display, but this is a small learning step.

3.3 Linking multiple displays of a single dataset

Linking displays is one of the most powerful paradigms in data visualization. In the simplest example, brushing or labeling cases in one display is immediately reflected in other displays, effecting a form of immediate graphical lookup across multiple attributes, variables, and dimensions.

In XGobi, linking requires inter-process communication and active trading of substantial amounts of data between processes.

In GGobi, however, linking is implemented with a simpler and more efficient paradigm based on data sharing: all displays have access to the same underlying data, hence linking can be implemented by sharing data regarding color, labeling or other presentation qualities. This scheme covers linking of displays that show the same data.

3.4 Linking multiple displays of multiple datasets

XGobi uses a "dirty" criterion for linking: it really doesn't know whether two displays are showing the same datasets or not: Roughly speaking, XGobi displays of different datasets are linked if they are of the same size. Linking is then performed by making the assumption that the datasets store the cases in the same order. This initial simple rule for linking later evolved into a rather complicated hodge-podge with special handling of "row groups," the "nlinkable" notion to exclude points from linking, and linking points to line segments (edges). In GGobi this has been replaced with a single set of rules that derive from a "clean" linking model for multiple datasets.

First, recall from section 2 that GGobi permits multiple datasets to be read from a single XML input file and to be viewed in the same process. It is also possible to add datasets sequentially, either interactively or through the API.

Now, GGobi establishes linking with one of two possible mechanisms: case identifiers or categorical variables. Either can be supplied in the XML input file.

- Linking by case identifiers

  The user can supply case or record identifiers as part of each dataset. Cases in different datasets will be linked if they share identifiers. This enables, for example, linking between displays of heterogeneous datasets that share only part of the cases, such as all cities of the Northeast in one dataset and the US cities with over 1 million inhabitants in another dataset.
- Linking by the values of a categorical variable

  Linking takes place according to the values of the categorical variable: if one case is touched (colored, "glyphed", labeled, ...) in one display, all cases with the same value of the categorical variable will change accordingly, in this and all other displays. For example, a categorical variable may classify cities into two types, "coastal" and "inland". If a coastal city is pointed at in one display, all coastal cities in this and all other displays will be highlighted. In this linking mode, displays of different datasets can be linked if they have a chosen categorical variable in common.

The major difference between case identifiers and categorical variables is that the former are unique within a dataset, whereas the values of a categorical variable are obviously not.

3.5 Some changes in view modes and tools

- The variable manipulation tool

  The table in the variable manipulation tool (seen in Figure 4) displays a few statistics for each variable: the current variable transformation (if any); the minimum, maximum, mean, and median of the raw data; the number of missing values per variable. This table can also be used to specify limits for variables or groups of variables.

  Variables can be added, too, using controls in this tool: an index variable representing the different colors and glyphs on the screen, or a variable containing the row numbers, or a copy of an existing variable.
- Color managers for brushing

  GGobi has a much richer notion of color selection than XGobi. Instead of a single fixed color scheme, GGobi offers a menu of choices: Users can specify one on the command line or by interacting with the "color schemes" tool. Most of the choices are derived from designs by Brewer (1999), whose work is based on color theory and tested in color perception experiments.

  Once a color scheme has been chosen, it can still be tuned: Double-clicking on any element of the color palette in the "color & glyph chooser" brings up a color selection widget with access to the full system color map (as seen in Figure 5). Notice that you can change the background color as well as any of the foreground colors.

- Panning and zooming interface for large data

  The direct manipulation panning and zooming methods work as they did in XGobi, but we've added what we call "Click-style interaction" for more precise control. The new interaction style is used to pan or zoom the plot by a fixed amount and is especially useful with big datasets, where the response of the plot to direct manipulation scaling can become sluggish.
- Sphering for projection pursuit and principal components

  In XGobi, when a user activates projection pursuit, all selected variables are automatically replaced by a set of principal components, but in GGobi, the principal components are added to the data as new variables. This allows them to be explored and compared to the untransformed data.
- Generalized tours

  The tour code in GGobi is completely redesigned, based on the algorithm in Buja et al. (1997) and the object-oriented tour code in Orca (Sutherland et al., 2000). The most obvious consequence is that there is now a 1D tour which generates a sequence of 1D projections and displays them as average shifted histograms. The two higher-dimensional modes, 2D tour and 2x1D (correlation) tour are still available, albeit with a smaller set of controls. The redesign brings the flexibility to experiment with other types of tours, for example, displaying 4D projections as parallel coordinate plots or scatterplot matrices. It also opens the components of the tour algorithm for access through the GGobi API.

4 Extending GGobi with an API

Too many statistical software projects are built from scratch and are not amenable to being used in existing environments. In many cases, much of the development effort is spent on creating incomplete or amateurish imitations of facilities in existing environments. Other projects are very tightly coupled to specific software such as R, S-Plus, MATLAB, SAS, and can be used in only one of these environments. Ideally, one would like to create software that embodies concepts and ideas in such a way that allows the software to be a) used as a stand-alone application, b) accessible from several of the statistical environments, and c) accessible from within existing or specialized software such as Excel, or in-house applications. In general, this is easy to do but requires consideration and planning at the initial stages of the software design.

GGobi has been developed in a way that allows its functionality to be used as a *C*-level library. The same routines and high-level data structures that are used in the stand-alone GGobi application are available to others who want to include GGobi in their own applications, and are also available as add-ons to languages such as R, Python, Perl, and any others that provide an interface

to C code.

While implemented in *C*, one can think of the GGobi library as providing an object-oriented class for creating GGobi instances, and methods for querying and manipulating each instance's state. The regular GGobi graphical interface uses these methods to implement the basic functionality. Different customized functionality and interfaces can be created by calling these same methods differently, and creating GGobi instances within other contexts and applications. We have developed an extensive interface between GGobi and R which allows one to use a high level programming language (S) to query and manage GGobi instances. We have also created interfaces to Perl and Python as existence proofs, and plan to use the ability to embed GGobi to provide direct manipulation, high dimensional visualization to Gnumeric, the Gnome spreadsheet.

From the developers' point of view, one advantage of creating embeddable software components is that experts can focus on what they're best at doing. The graphics experts on the GGobi development team can continue to focus on graphical methods and the user interface, and they do not need to devote any effort to creating yet another specialized, ad hoc scripting language. Instead, GGobi can leverage existing and familiar languages. Additionally, GGobi can avoid many data management issues, random number generation details, etc. and exploit the expertise and well-tested code of others in these areas.

This approach has other benefits for developers and users of the host applications. Specialized software created as embeddable components can be used in other applications and contexts, with its functionality encapsulated in an API so that details are hidden. Embedding the software in a variety of host applications makes it available to a wide audience via different and specialized interfaces. R users who prefer the command-line interface can exploit much of GGobi's functionality without using its direct manipulation interface. Devotees of the graphical user interface can continue to use GGobi without using any written language at all.

Another advantage (for GGobi developers *and* users) of providing GGobi as a library as well as a stand-alone application is that some functions can be omitted from the stand-alone application (protecting the GUI from needless clutter) and still provided in the library. This is especially useful for tasks that are not well suited to a GUI. For example, one feature that is available through the API but not in the GUI is the ability to create mixed or composite displays. In other words, we can create a new display that may contain any number of any kind of plots, and control how those plots are laid out within the window. In R, the function **plotLayout()** allows one to specify a collection of plot descriptions along with the cells in a grid that each should occupy to create a new display. This provides a way to arrange the plots in different

configurations that best display the features of interest, enhancing the visual presentation of linking, etc. This is too complex to provide in a simple GUI and may be of more use to advanced users.

This style of development is reasonably standard in the software engineering and design communities, but has not been adopted widely in the statistical computing world. It has been explored and used to good effect in the Omegahat project and subsequently in R, allowing these environments to be used transparently in many varied settings such as within databases, Web browsers, and spreadsheets. More details can be found in Chambers (2000) and Temple Lang (2000).

The embeddable approach works very well when the two software projects are a) well established and have a reasonable following and active community of users and developers, and b) are Open Source, allowing greater flexibility for users to explore different ways to use the software.

4.1 Evolution

The designers of XGobi had always intended that it be used in conjunction with some analytical software. It is instructive to examine previous efforts towards this aim. There exists an S (i.e. R or S-Plus) function, distributed with the XGobi software, that allows an S user to launch an XGobi process given S objects as arguments. That function is embarrassingly simple: the S objects are written out as ASCII files, and a system call executes XGobi with those files as arguments. An XGobi process launched in this way has very limited ability to create S objects directly: after brushing, for example, the vector of point colors can be saved as a file in the S format. The S process has no ability to communicate further with XGobi.

The XGobi authors occasionally explored other approaches that would extend this unsatisfactory relationship (Swayne et al., 1991; Symanzik et al., 1999). As early as 1991, we used interprocess communication to maintain a live connection between XGobi and S. One of the applications would draw a clustering tree in S, allow the user to click on it to cut the tree and immediately set the point colors in XGobi to show the result. It relied on a second program, also written in C, to gather input from the user and to manage the interprocess communication. It was necessary to assemble each command in the S language, ship it to S, read back the result and respond accordingly. To make this foolproof, it would have been necessary for XGobi to be fully able to parse S commands and handle errors. Because this was such a daunting task and one that would require continual updates to keep pace with changes to the S language, work on this model was discontinued after the first prototype.

Whenever interprocess communication is used, synchronizing communication between the two processes can be a non-trivial issue. Higher level synchronization – notifying one process of changes made to the data in the other – is outside of the capabilities of these models but is important for a general interface.

The R-GGobi interface is implemented as a regular R package that one can choose to use at different points in an R session. The result is that there is a single R process and one or more GGobi instances that run within the process. The relationship doesn't need to use interprocess communication, because there's only one process. There's no need to write a GUI to assemble each command in the S language, because the user will supply it directly to R in the customary way. Parsing of input and output between the two systems is also unnecessary as values are passed directly between them.

The only complexity resulting from embedding follows from the way user events in GGobi are handled by the host application, e.g. R. To handle input for R from the R graphics devices and console and simultaneously recognize events in GGobi, R needs to support multiple input sources. Work to generalize the R event loop has made this possible. However, more work is needed and research into the use of multiple threads within R is under way.

4.2 The GGobi API

The GGobi API allows the basic facilities of the stand-alone application to be incorporated into other applications, and to be made available in other programming languages such as R, Perl, Python, and C. The routines in the API provide access to the entire GGobi computational model. It allows us to create new GGobi instances and manipulate both the data and the contents of the displays. The routines allow us to both query and set the state of multiple GGobi instances within the same process and also provide custom event handlers for certain high-level interactions.

The routines in the API allow us to create new GGobi instances at any time, specifying data either directly from memory (in the form of a data frame or matrix), or externally from a file or URL. We can access the data from the new GGobi instance and also replace individual cells. We also add or remove variables from a dataset, edit the variable names, and generally retrieve and modify the data.

We can retrieve and set the visual characteristics of points in the display – the color, glyph type and size of each point – and choose whether a point is displayed or not. We can query the current status of the active brushing region, including its position, color and even the points it encloses. While brushing

and point identification are usually done interactively, GGobi's brushing region can be controlled programmatically. This can be appealing when the number of points becomes large, because the brush response begins to lag behind the user's hand. It is also convenient for providing directed animations or "movies" and also allows one to provide scripts which create a particular view that one wants a user to see.

As mentioned earlier, a GGobi process can have several plot displays open at the same time, and there are several types of displays, such as scatterplots, scatterplot matrices, and parallel coordinates plots. The API provides ways to manipulate the set of displays. For example, we can determine what displays are open, delete a particular display, open a new one, and control which one is active.

4.3 Embedding for tighter coupling

The facilities described above allow applications (e.g. R) to communicate with GGobi and both set and query its state. This, in itself, is a significant improvement over the previous approaches. However, it still preserves a separation between the GGobi and the host application. A richer form of embedding allows the two systems to interact in a more dynamic and symbiotic manner. Two examples illustrate this. First, GGobi can use functionality in the internal R API. For example, the random sampling of records provided by GGobi can make use of the random number generation facilities in the R math library. In an experimental version of GGobi, smoothed values can be displayed on a plot. The interface between GGobi and R allows users to provide an S-language function to perform that smoothing, updating the points on the plot as the user moves the GGobi slider to change the smoothing parameter(s). This allows new smoothing functions to be quickly explored in the direct manipulation world.

More interestingly, we can allow user-level S-language functions to be specified at any time within the lifetime of a GGobi session to implement particular tasks. For instance, users can specify functions that will respond to a GGobi event. As an example, let's consider GGobi's *identify* mode for interactively labeling points. As the user moves the pointer close to a point on the plot, the label for that case is displayed. The identification of a point corresponds to a high-level GGobi event and users can specify an S function which is to be called when such an event occurs. In this case, the function is given the index of the observation and can use this, for example, to highlight the corresponding row in a data editor or to display some characteristic of the observation in an R plot. This allows users to easily customize how GGobi responds to a growing number of user interactions using a high-level programming language, S.

Another form of event handling allows one to associate a function in R with one of the number keys (i.e. 0, 1, ..., 9). When that key is pressed while viewing a GGobi plot, the function is called. This can be used for tasks like computing summary statistics or updating a plot based on the current state of the GGobi display.

This type of event-driven, dynamic feedback to the high-level programming language makes it significantly easier to experiment with and develop interactive graphics tools. It is relatively easy to implement customized linking algorithms and explore characteristics of different statistical methods with simple user actions. Programming these in the low-level GGobi *C* code is prohibitively complex. But exposing these top-level events as programmable actions for high-level scripting languages significantly reduces the cost of experimentation.

5 Plugins

Plugins are a modular mechanism by which developers (both the original authors of GGobi and others) can add extensions and optional features to GGobi without interfering with the existing code base or adding to the complexity of the code and configuration process. In this way, developers who are willing to learn about the GGobi internals can add more substantial extensions to GGobi than would be possible if they restricted themselves to the use of the API.

Plugins allow different installations of GGobi to provide different features depending on the goals of the users and the availability of third-party libraries on the machine at that site. The example plugin provides a data grid or spreadsheet-like layout for displaying datasets and their values in a GGobi instance. It uses the `GTK+Extra` library which is not as commonly installed on machines as the standard `GTK+` libraries. If this library is available, one can compile and install the data grid plugin for GGobi. Users will then see an additional menu option in the GGobi control panel and can view and (eventually) edit the data.

Additional plugins that we might explore include facilities to read and write data residing in different sources such as relational database servers, and data in proprietary formats and specialized XML formats. We might also write plugins to generate plots in various formats such as Postscript, PDF, or SVG (Scalable Vector Graphics, an XML-based graphics representation).

The plugin facility could become "multi-lingual." It is possible to implement the functionality of the plugin in languages other than *C*. For example, we

might provide graphical interface plugins using Perl's and Python' `GTK+` bindings, simplifying the development of such plugins. One can also embed the Java virtual machine inside GGobi and provide plugins via arbitrary Java classes.

The plugin system is similar to that used in Gnumeric, the Gnome spreadsheet application. It uses XML to describe and identify the plugin to the GGobi process. One can control which plugins are available by editing the XML file, and one can activate and deactivate the different plugins during the GGobi session via the GGobi GUI.

6 Using GGobi: An Example

The data used in the example comes from a study of gene expression during the development of the central nervous system of the rat (Wen et al., 1998). There are 112 records, each representing the expression level for a single gene measured at nine time points, from embryonic day 11 to adulthood, defined as post-natal day 90. There are two additional variables, indicator vectors that map each gene into one of four functional classes and 14 sub-classes.

We'll start by exploring the data in GGobi alone. We initiate GGobi and we see the result in Figure 3: the GGobi console window and a single scatterplot.

We conduct a a preliminary exploration of the data, starting by opening the variable manipulation panel shown in Figure 4. The variables representing the time points are named *E11*, ..., *E21* for embryonic days 11, 13, 15, 18 and 21, and *P0*, ..., *P14* for postnatal days 0, 7 and 14, and finally *A* for adulthood. The range of each of these variables is $[0, 1]$, indicating that they have been been standardized. We look at a few scatterplots, selecting new variables by clicking in the variable selection panel. It is not surprising that measurements taken closer in time often appear to be highly correlated, while those taken more than one or two time steps away from each other do not look correlated at all. We do not immediately see any outliers that provoke our curiosity.

What we are curious about is whether the four functional classes represented by the variable *Class1* reveal distinct patterns in the progress of gene expression during the study period.

We start by brushing the points according to the functional class variable. We open a tool called the "color & glyph chooser" (introduced in section 3.5, and shown in Figure 5) to select colors and symbols for brushing. We select a different color (and a different glyph, since these graphics will be probably be published in shades of gray) for each of the four functional classes, and brush

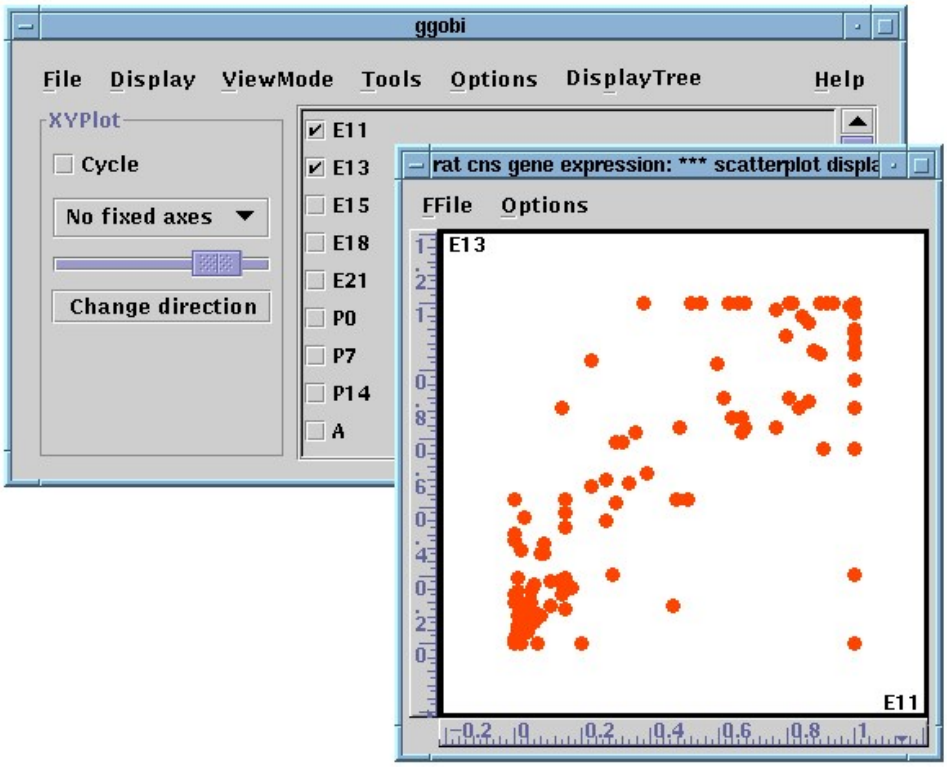

Fig. 3. The GGobi console window and a scatterplot of the 112 gene expression levels at the first and second time points, with the levels at embryonic day 13 (*E13*) plotted against the levels measured two days earlier.

the points.

To examine each of the classes in turn, we use a GGobi tool designed for just this purpose: the rightmost window in Figure 6, labelled "color & glyph groups." It has one column for each group of points with a unique combination of color and glyph. Its first column shows the group's symbol, and its last three columns report the number of points hidden and shown. Clicking on the *H* or *S* button hides or shows all the points in the group.

The display we choose for looking at each class in turn is the parallel coordinates plot (Inselberg, 1985; Wegman, 1990), which is particularly useful for

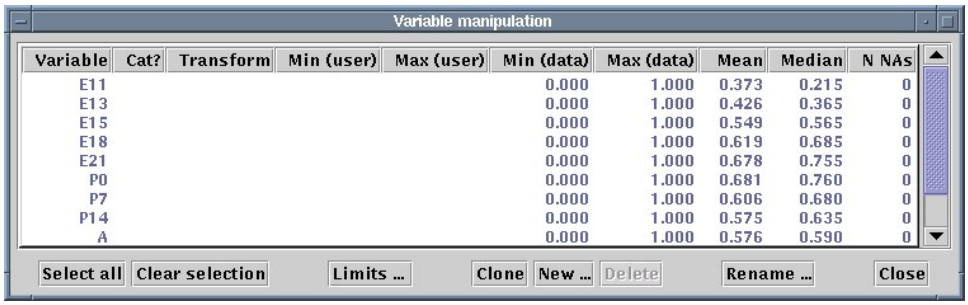

Fig. 4. The GGobi variable manipulation panel, showing the range, mean and median for each variable.

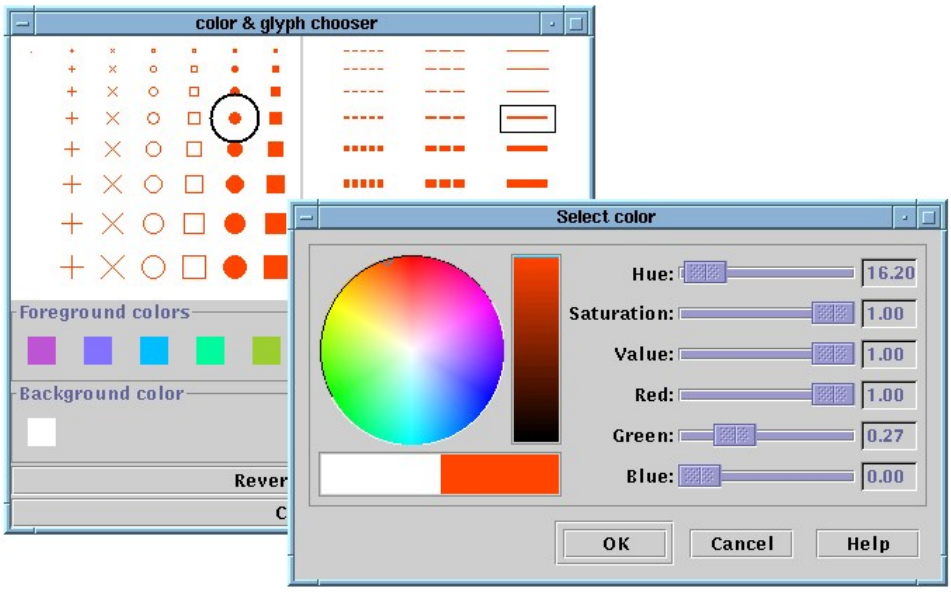

Fig. 5. GGobi's panel for choosing color, glyph and line type, together with the GTK+ color selection widget which allows any color in GGobi's colormap to be edited.

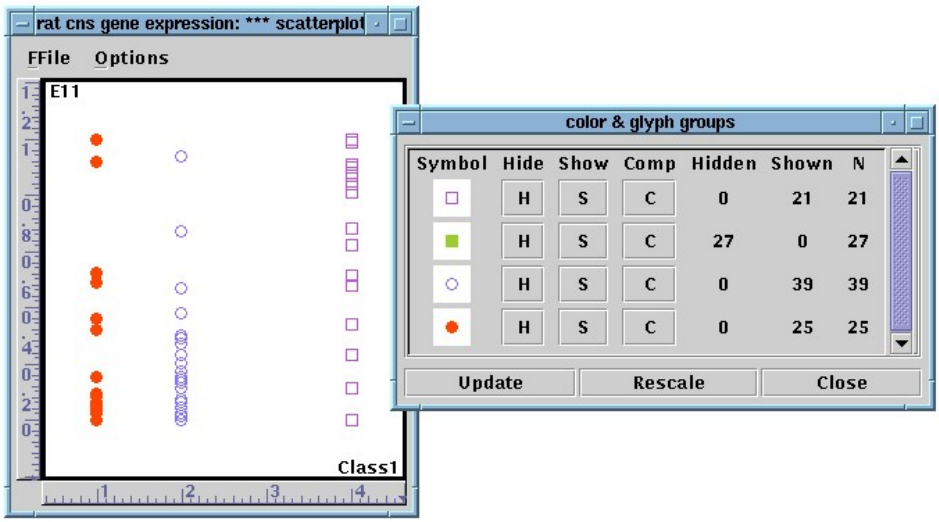

Fig. 6. A scatterplot with the expression levels on embryonic day 11 plotted vertically and the functional class variable plotted horizontally, and GGobi's tool for selectively hiding groups of points with a common glyph and color. Of the four functional classes, the third has been hidden by clicking on the corresponding **H** button in the tool.

these data because the variables are measurements on the same scale and in time order. The resulting display approximates a time series plot, although the time steps are not uniform. Instead of hiding one class, as shown in the scatterplot in Figure 6, we hide three and show only one. The resulting parallel coordinate plots are shown in Figure 7. (These four plots were not on the screen at the same time, but rather represent a sequence of plots we generated using the "color & glyph groups" tool.)

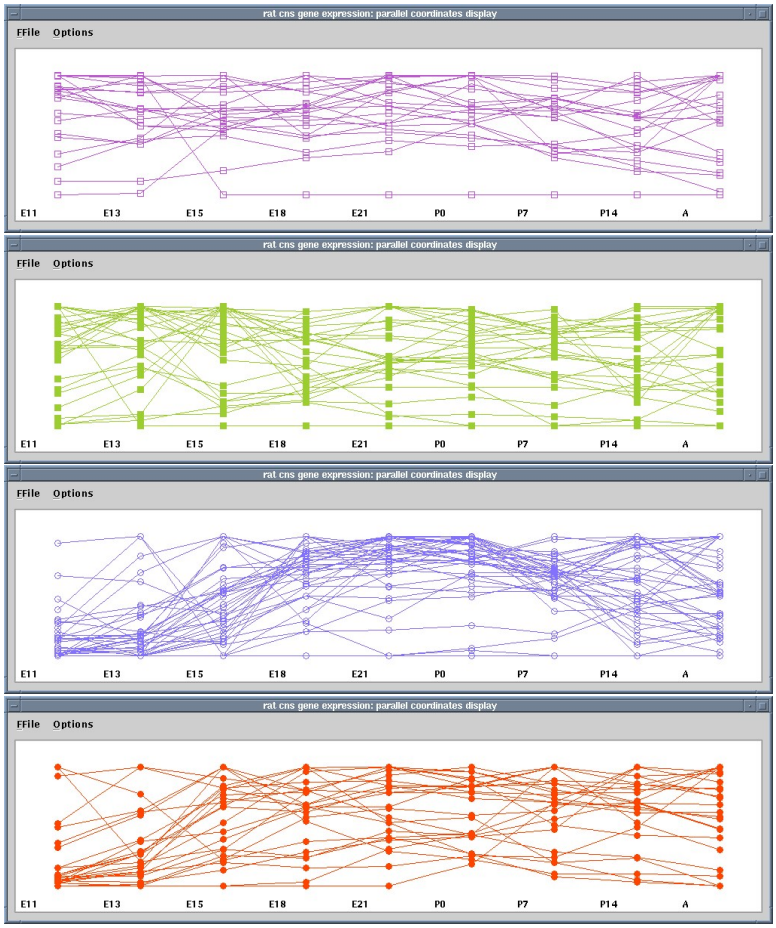

Fig. 7. Four parallel coordinate plots, each of which shows all the cases corresponding to one of the four functional classes. We generated these plots one at a time using the tool shown in Figure 6 to control which groups are shown and which are hidden.

While these functional classes clearly have some structure, we wonder if we could find a formal clustering of the data that would yield more coherent groups. To answer that question, we use the link between R and GGobi. We start R, then load the necessary R package and initiate GGobi using the same XML file that we used when we started GGobi running on its own.

```
> library(Rggobi)
> ggobi ("ratcns")
```

To perform the hierarchical clustering, we read the data from GGobi into R, find a distance matrix and generate the dendrogram. Since we can use GGobi and the standard R graphics devices at the same time, we use the R to display the dendrogram, as shown in Figure 8. We use the R function *cutree* to cut the tree so as to produce five clusters and to create the vector *cvar* which consists of the cluster identifiers for each case. Finally, we add the new variable to GGobi using one of the functions in the Rggobi library. As we add the variable, its label appears in the variable selection checkboxes and a new

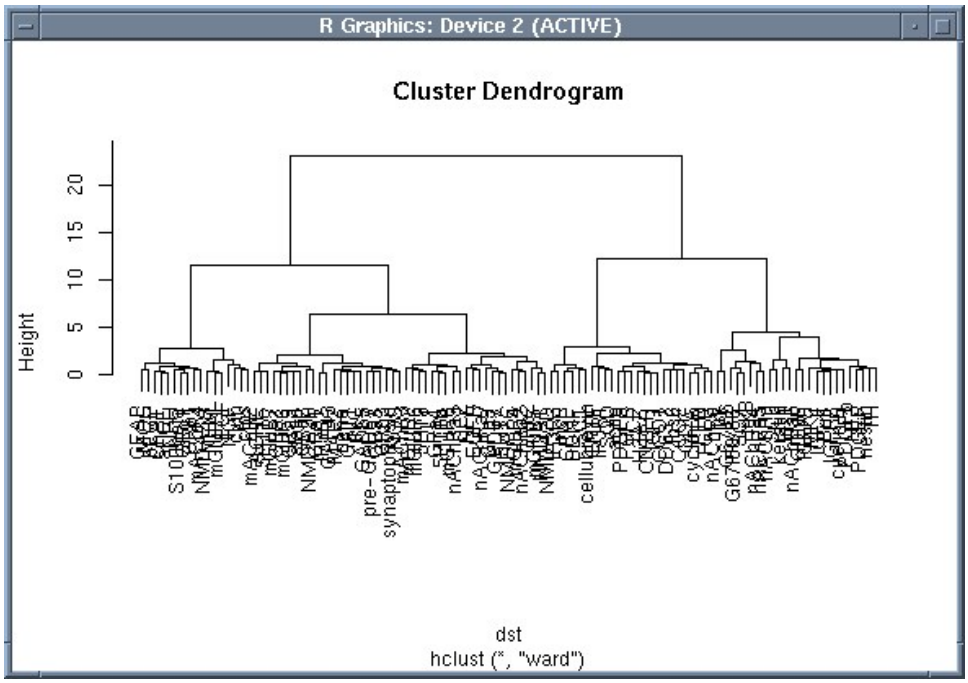

Fig. 8. The dendrogram of the genes in the study, produced using R's *x11* driver. We will cut the tree so as to produce five clusters.

row is added to the variable manipulation tool.

This is the associated R code:

```
> ratcns <- getData.ggobi()
> dst <- dist(ratcns[,1:9])
> hc <- hclust(dst, "w")
> x11()
> plot.hclust (hc)
> cvar <- cutree (hc, 5)
> addVariable.ggobi (cvar, "cvar")
```

6.1 Animation

Since the variable of cluster identifiers has been added to the GGobi process, we could use brushing to compare the five clusters, first interactively erasing all points not in the first cluster (i.e., those points for which the value of *cvar* is not equal to 1), then all points not in the second cluster, and so on. Since we are working with R, though, we can compare the clusters by writing a simple script to animate through them, then sit back and watch it run.

First we write a small function to show only those cases for which the value of the vector v is equal to n:

```
> showCases <- function (n, v) {
>    tmp <- logical( length(v) )
>    if (n != -1) {
>      tmp[v != n] <- T
>    }
>    setHiddenCases.ggobi(tmp)
> }
```

Then we loop through *cvar*, showing one cluster at a time, with a 2-second pause between plot changes:

```
> for (i in 1:5) {
>   showCases (i, cvar)
>   system("sleep 2")
> }
```

The plots we cycle through are shown in Figure 9. Comparing this figure with Figure 7, we conclude that most of the groups produced by hierarchical clustering are more homogeneous than the functional classes. If you could see the plots in color, you would see that we have retained the colors that correspond to the gene's functional classes, which also allows us to see which classes are represented in each cluster.

6.2 Event handling

GGobi events can trigger the execution of functions in R, as described in Section 4.3. During GGobi's *identify* mode, as the mouse is moved around the screen, the point nearest the cursor is labeled. At the same time, an event is generated, and we can have R respond to it. We first define the function we want to have executed; its first argument is the index of the labeled point. We then set that function to be the identify event handler. In this example, we simply print the expression values for the indicated gene:

```
> f <- function (i, .ggobi) {
>   if (i >= 0) {print(ratcns[i+1,1:9])}
> }
> setIdentifyHandler.ggobi (f)
```

As we move the cursor across the screen, the points in the GGobi display are labeled as before, but we also see an immediate response in the R window, in which the gene expression values are displayed.

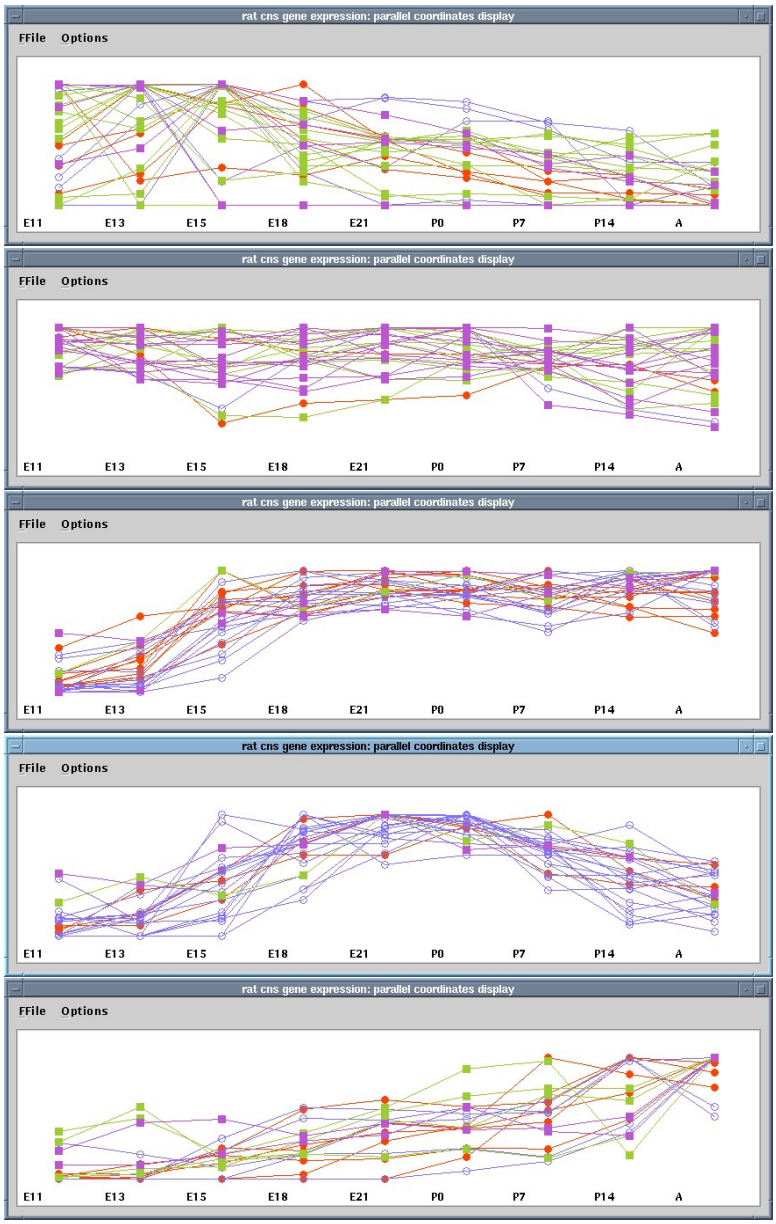

Fig. 9. Five parallel coordinate plots, each of which shows all the cases corresponding to one of the five groups found by using hierarchical clustering in R. These are exactly the views that we animate by executing a loop in R.

7 Conclusions

This smooth integration of R and GGobi is an example of a modern approach to component-based software inter-operability in statistical computing. No single piece of software has to do everything, but any piece of software can be designed to work well with others. GGobi's design was greatly changed so that it could be compiled into a library and controlled via an API; R was changed so that it could accommodate GGobi's event loop along with its own.

This software combination may hold special interest for teachers of statistical computing courses. If R or S is already on the syllabus, this environment should make it easier to introduce interactive graphics, and it should also be a good platform for student projects.

We also hope that this combination will encourage other software developers to pursue a component-based strategy instead of "reinventing the wheel" whenever they need a bit of new functionality. By relying on existing software, and designing with that software in mind, new tools can be created that are more powerful, more reliable and more useful than if they are created in a vacuum.

We should not forget to mention that the GGobi software and documentation, including sample scripts in the S language, are available on the web site *www.ggobi.org*.

References

Becker, R. A., Chambers, J. M., Wilks, A. R., 1988. The New S Language: A Programming Environment for Data Analysis and Graphics. Wadsworth & Brooks/Cole, Pacific Grove, CA.

Brewer, C. A., 1999. Color use guidelines for data representation. In: Proceedings of the Section on Statistical Graphics. American Statistical Association, Baltimore, pp. 55–60.

Buja, A., Cook, D., Asimov, D., Hurley, C., 1997. Dynamic Projections in High-Dimensional Visualization: Theory and Computational Methods. Tech. rep., AT&T Labs, Florham Park, NJ.

Chambers, J. M., 2000. Users, programmers, and statistical software. Journal of Computational and Graphical Statistics 9 (3), 404–451.

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

Inselberg, A., 1985. The Plane with Parallel Coordinates. The Visual Computer 1, 69–91.

Sutherland, P., Rossini, A., Lumley, T., Lewin-Koh, N., Dickerson, J., Cox, Z., Cook, D., 2000. Orca: A Visualization Toolkit for High-Dimensional Data. Journal of Computational and Graphical Statistics 9 (3), 509–529.

Swayne, D. F., Buja, A., Hubbell, N., 1991. XGobi meets S: Integrating software for data analysis. In: Computing Science and Statistics: Proceedings of the 23rd Symposium on the Interface. Interface Foundation of North America, Inc., Fairfax Station, VA, pp. 430–434.

Swayne, D. F., Cook, D., Buja, A., 1992. XGobi: Interactive dynamic graphics in the X Window System with a link to S. In: American Statistical Association 1991 Proceedings of the Section on Statistical Graphics. American Statistical Association, Alexandria, VA, pp. 1–8.

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.

Symanzik, J., Cook, D., Lewin-Koh, N., Majure, J. J., Megretskaia, I., 1999. Linking ArcView 3.0 and XGobi: Insight Behind the Front End. Journal of Computational and Graphical Statistics 9 (3), 470–490.

Temple Lang, D., 2000. The omegahat environment: New possibilities for statistical computing. Journal of Computational and Graphical Statistics 9 (3), 423–451.

Wegman, E., 1990. Hyperdimensional Data Analysis Using Parallel Coordinates. Journal of American Statistics Association 85, 664–675.

Wen, X., Fuhrman, S., Michaels, G. S., Carr, D. B., Smith, S., Barker, J. L., Somogyi, R., 1998. Large-scale temporal gene expression mapping of central nervous system development. In: Proceedings of the National Academy of Sciences of the United States of America. Vol. 95. Washington, DC, pp. 334–339.