DSC 2001 Proceedings of the 2nd International Workshop on Distributed Statistical Computing
March 15-17, Vienna, Austria
http://www.ci.tuwien.ac.at/Conferences/DSC-2001
K. Hornik & F. Leisch (eds.) *ISSN 1609-395X*

Orca [R [RJava]]

Thomas Lumley*

Abstract

Orca (`http://software.biostat.washington.edu`) is a project to develop and implement dynamic graphics for displaying high-dimensional data, particularly time series, spatial, and network data. Integrating these graphics into a statistical environment will simplify data manipulation, allow statistical modelling to assist with graphical display and graphical displays to assist modelling. I briefly describe the design of Orca and two ways to link it to R.

1 Introduction

Orca (`http://software.biostat.washington.edu`) is a project to develop and implement graphics for displaying high-dimensional data, particularly in cases where the physical context of the data is important[3]. For example, in understanding high-dimensional spatial data both the relationships between variables and the spatial patterns of these variables are potentially important. Software such as XGobi ([4]) can display high-dimensional data and, conversely, GIS and scientific visualisation tools can display single variables in maps, time series or animations. When these goals are combined there is relatively little research into what to display or how to display it.

One of the basic graphical tools used in Orca is the Grand Tour and its variations. The Grand Tour[1] is based on the observation that a distribution can be characterised by its projections into low-dimensional subspaces. A sequence of scatterplots is shown, based on a two-dimensional projection of the data that rotates through the higher-dimensional space. The original version used irrational rotations to create a dense set of projections. Current implementations use random

*Orca was developed with partial funding from the US EPA

```
public TaoExample (String filename, String groupname,
                           String timename) {
   OrcaData od = org.orca.data.parsers.OrcaDataSource.openFileData(filename);

   od = new MarkTimeSeriesData(od, timename);
   od = new MarkGroupData(od, groupname);

   DataSourcePipe dsp = new DataSourcePipe(od);
   StdDataPipe stdd1 = new StdDataPipe(dsp);
   ReorderPipe stdd=new ReorderPipe(stdd1);

   MultiLayoutPipe mlp = new MultiLayoutPipe(stdd);
   WindowPipe obw = new WindowPipe(mlp);

   TourPipe ortp1 = new TourPipe(stdd);
   Render2DPipe t2p2 = new Render2DPipe(ortp1);
   WindowPipe obw4 = new WindowPipe(t2p2);

   TourPipe ortp2 = new TourPipe(stdd);
   TimeSeriesPipe t2p = new TimeSeriesPipe(ortp2);
   WindowPipe obw3 = new WindowPipe(t2p);

   obw.reloadView(); obw3.reloadView(); obw4.reloadView();
   }
```

Figure 1: A time series tour

rotations, often guided by stochastic optimisation of some criterion that selects potentially interesting projections.

2 Orca

Orca is designed as a toolkit that can be easily extended and used with a little knowledge of Java. It is made up of objects that can be assembled into a 'pipeline' processing structure. The structure itself is not new, having been described by [2]. The easy extensibility is new, and is based on the use of Java interfaces to define all the required relationships between pipeline sections.

Figure 1 shows the Java code for constructing three plots of a multi-group multivariate time series: a scatterplot matrix, a two-dimensional random Grand Tour, and a time series plot where the vertical axis tours through one-dimensional projections of the data. This was used to display data from two stations in NOAA's Tropical Atmosphere Ocean experiment monitoring climate in the tropical South Pacific ocean.

The first step is to read in the data from a file. The time and group (monitoring

station) variables are marked as describing aspects of the sampling rather than measured variables. A sequence of objects then process the data to standardise the variables and order by time. These may or may not result in copying of the data: each pipe section has a method that returns the processed data, but these data can either be stored or be obtained from the previous pipe section and reprocessed on the fly. In addition to passing along data, each section can add controls or navigation displays for the final plot. The interface specification looks like

```
public interface OrcaPipe {
  // methods for passing data through pipeline
  public OrcaControl getControls(OrcaPipe op);
  public OrcaAppearance getAppearance(OrcaPipe op);
  public OrcaData getViewData(OrcaPipe op);
  public Object getNavigation(OrcaPipe op);

  // methods for linking pipes and notifying of changes
  public void propagateChangeForward(OrcaEvent oe);
  public void propagateChangeBackward(OrcaEvent oe);
  public void couplePipe(OrcaPipe op);
  public void replacePipe(OrcaPipe op);
  public void decouplePipe(OrcaPipe op);
  public void removeOutPipe(OrcaPipe op);
  public void addOutPipe(OrcaPipe op);
}
```

The two tour plots have a section that creates random projections. The three plots then each have a pipe section that computes the points to be displayed an their coordinates on the screen, and a final section that does the actual drawing.

The use of interfaces makes it easy to create a new section (the `TimeSeriesPipe` was written in about a day) and to create more or less complicated sets of plots. For example, the generation of a new projection is done by calling the `generateBasis` method of a `Tour` object. Adding guided tours just meant replacing the random-direction object by a more sophisticated one. This required recompiling, but the next step would be to write a method that allows a new `Tour` object to be supplied at any time.

The pipeline structure gives linking ability for free. If brushing information is stored in the `DataSourcePipe` it will immediately be shared among all the windows descended from that data source. Similarly, if two plots are rendered from the same `Tour Pipe` they will show the same projections of the data.

As it stands the software requires writing and compiling a program to change the displays. This is clearly undesirable, especially if it is to be linked to interactive statistical modelling. The graphs can be created interactively in any Java interpreter (we have used `beanshell` and `JPython`), but this still does not give the desired statistical functionality. As Omegahat develops it will provide an alternative, but for the present we need to link to an available statistical package.

3 Linking to R

In order to run Orca from R we need two things: the ability to issue Java commands and receive input from Java, and a way to set up R objects that mirror the Orca pipe sections. I will describe two ways of doing this. The first is inelegant, slow, inconvenient to program, insecure and reliable. The second is elegant, slow, more convenient, more secure and currently less reliable.

3.1 Connecting to R via beanshell

At the first DSC I described, but did not demonstrate due to hardware problems, a link between R and XLISP-Stat using sockets. I had modified Luke Tierney's socket code for XLISP-Stat and loaded it into R, and had an R histogram controlled by XLISP-Stat sliders rather along the lines of R's current `demo(tkdensity)`. The Java interpreter beanshell (`www.beanshell.org`) is a lightweight Java interpreter designed for prototyping and scripting. For our purposes the most important feature of beanshell is that it can be run from a socket connection instead of a text console, and so can be controlled using R's socket functions.

The R interface involves a function closure that contains a reference to a socket connection to beanshell. This function sends its text argument to beanshell and returns the text response, with the option to block until a particular pattern is seen (in particular, the beanshell prompt), roughly the way ESS communicates with its statistics packages.

A small sample from `OrcaTimeSeriesTour()` shows how the correspondence between R and Java objects is set up. Here `orca()` is the connection object and `newOrcaName()` uses `tempfile()` to create unique names.

```
dname1<-newOrcaName()
orca(paste("DataSourcePipe",dname1,"= new DataSourcePipe(",dname,");"))
dname2<-newOrcaName()
orca(paste("StdDataPipe",dname2,"= new StdDataPipe(",dname1,");"))
```

The communication is two-way, specifically we have functions `brushing` and `brushing<-` that read and write the brushing colors from an Orca object. The following code is from `example(OrcaSpatialTour)`, which plots a surface whose height is a one-dimensional tour of a dataset. The example uses the dataset on US States, with Alaska and Hawaii removed

```
statetour<-OrcaSpatialTour(sd,x="x",y="y")
# color WA red, CA green to help orient map
brushing(statetour)<-cbind(c(4,45),c(7,16))
#color by illiteracy level
brushing(statetour)<-23-round(sd$Illiteracy*3)
```

As the example shows, the brushing information can be written (and read) either by specifying index and value or by giving a complete vector of new values.

The limitations of a socket connection with the objects stored on the far side make reading information from Orca somewhat cumbersome. For example, the major work in `brushing()` is done by

```
loop<-paste("for(int",j,"=0;",j,"<",dot(i,"length"),";",j,"++)
                  {print(",subs(i,j),");}\n print(\"*\");")
r<-orca(loop,read.until="\\*")
s<-strsplit(gsub("bsh %","",r),"\n")[[1]]
ii<-grep("\\*",s)
n<-as.numeric(s[1:(ii-1)])
```

which creates a `for` loop in Java to print out the color numbers and then parses the resulting string in Java.

3.2 Connecting with RJava

Duncan Temple Lang has produce a large collection of links from R to other languages for the Omega project (`www.omegahat.org`). Naturally, one of these is to Java. The highly threaded nature of Java doesn't fit easily with R, and Java itself is not completely stable, especially on Linux. However, the link to Orca now appears to work if the instructions are followed precisely. The RJava-Orca link is largely the work of Tony Rossini, who was probably the first person outside New Jersey to get RJava running consistently.

The follow code shows the setup for the time series tour part of the code in Figure 1.

```
dsp.1 <- .JavaConstructor("DataSourcePipe", od.1,.name="dsp")
stdd1.1 <- .JavaConstructor("StdDataPipe",  dsp.1, .name="stdd1")
stdd.1 <- .JavaConstructor("ReorderPipe", stdd1.1, .name="stdd")
## Set up the TimeSeries
ortp2.1 <- .JavaConstructor("TourPipe",stdd.1,.name="ortp2")
t2p.1 <- .JavaConstructor("TimeSeriesPipe",ortp2.1,.name="t2p")
obw3.1 <- .JavaConstructor("WindowPipe",t2p.1,.name="obw3")
```

For simply setting up the plots there is little difference between this and the socket interface. The advantage of the RJava interface will come with integrating Orca into statistical modelling. With the socket interface the Orca objects are accessible to R only by issuing commands to write out the desired information as text. This text then has to be parsed by R. Using RJava the Orca object methods can be called directly from R and can return complex objects.

Ironically, the difference may become less important with another development initiated by the Omega project. If R and Orca can communicate in XML then quite complex information can easily be transferred over a single text channel, and other concerns such as speed and portability may become relevant.

4 Further development

In addition to some more development of the interfaces, further integration of Orca with R requires the more difficult problem of deciding what pictures to display and what computations they need. The Orca project has always recognised that

implementation is the easy half of what we are trying to do. We hope that providing better interfaces will encourage other people to try Orca and have ideas about new uses for dynamic graphics.

References

[1] D Azimov. The grand tour: A tool for viewing multidimensional data. *SIAM Journal of Scientific and Statistical Computing*, page 6, 128–143 1985.

[2] A. Buja, D. Asimov, C. Hurley, and J. A. McDonald. Elements of a Viewing Pipeline for Data Analysis. In W. S. Cleveland and M. E. McGill, editors, *Dynamic Graphics for Statistics*, pages 277–308. Wadsworth, Monterey, CA, 1988.

[3] Peter Sutherland, Antony Rossini, Thomas Lumley, Nicolas Lewin-Koh, Julie Dickerson, Zach Cox, and Dianne Cook. Orca: A visualization toolkit for high-dimensional data. *Journal of Computational and Graphical Statistics*, 9:509–529, 2000.

[4] Deborah F. Swayne, Dianne Cook, and Andreas Buja. Xgobi: Interactive dynamic data visualization in the X Window System. *Journal of Computational and Graphical Statistics*, 7:113–130, 1998.