xmapbridge: Graphically Displaying Numeric Data in the X:Map Genome Browser

Tim Yates, Crispin J Miller

October 14, 2015

Contents

1 Introduction

xmapbridge is a package that allows numeric data to be displayed in the web based genome browser, X:Map. X:Map uses the Google Maps API to provide a real-time scrollable view of the genome. It supports a number of species, and can be accessed at `http://xmap.picr.man.ac.uk`.

To do this, it communicates with X:Map via a small Java application called *XMapBridge*. *XMapBridge* sits on your local machine, interprets the graph data generated by *xmapbridge* and uses it to create plots, which are then passed to the genome browser to be displayed via HTTP.

Importantly, all graph rendering is performed on the local machine so none of the underlying graph data is uploaded to the X:Map server to generate your graphs.

Graph plotting in R is done using calls to the functions `xmap.plot` and `xmap.points`, which have parameters that aim to be similar to those used by the standard `plot` methods in R. These result in data being written to a set of files (in a specific directory

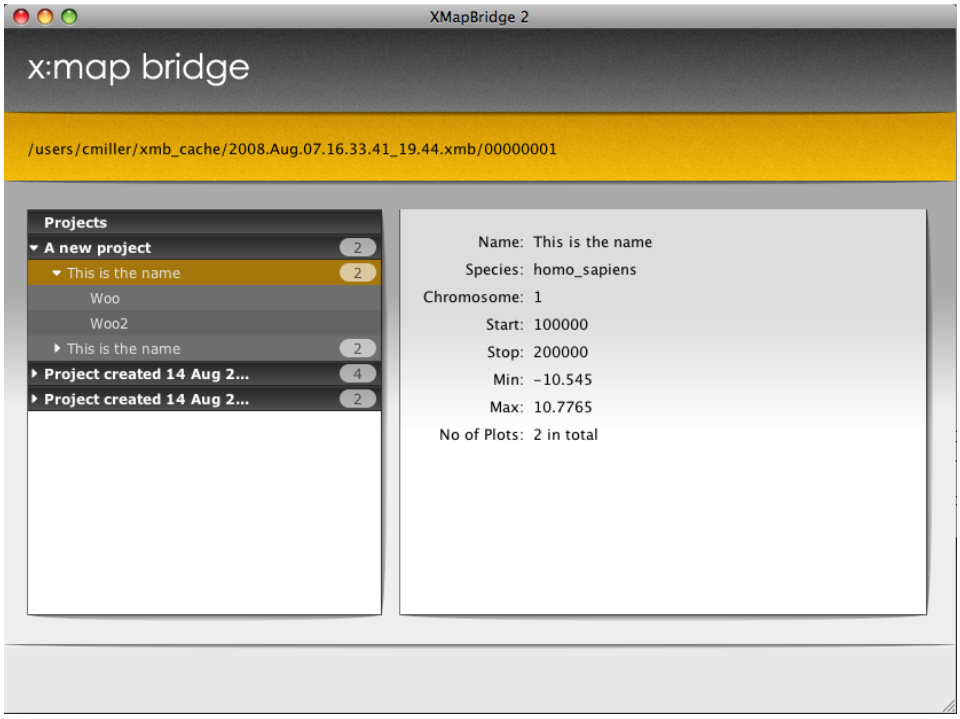

Figure 1: The XMapBridge in operation

structure) that contain the data to be displayed, as well as some additional meta-data describing each of the graphs.

When the *XMapBridge* application is running (figure 1), it monitors the same directory structure used by *xmapbridge* and uses the information written out from the Bioconductor session to generate its plots. These are then passed via HTTP to the genome browser for display.

> Note that the *XMapBridge* must be running on the same machine as your webbrowser. Only one instance of the XMapBridge can be run on a machine at any one time, and the X:Map webpage will try to connect to localhost to load javascript and graph data. If you do not see the "Connect" button when *XMapBridge* is running, the most likely reason is this.

2 Quickstart

The steps required to get the *xmapbridge* and *XMapBridge* programs working are as follows:

1. Install Java 5+ (if it's not already installed).
2. Install *XMapBridge* (see the section 'Installation of the XMapBridge software' at the end of this Vignette for more details).
3. Install *xmapbridge* (this package).

By default, the *XMapBridge* java application and the *xmapbridge* R package use a folder called *.xmb_cache* in your home directory.

If this folder does not exist when the R functions are first called (or the environment variable isn't set – see below), then *xmapbridge* will offer to create the folder for you.

If this folder does not exist when the *XMapBridge* is first run, (and there is no environment variable set), then the application will run in demo mode, showing three demo graphs for you to see how it works on your system (figure 2).

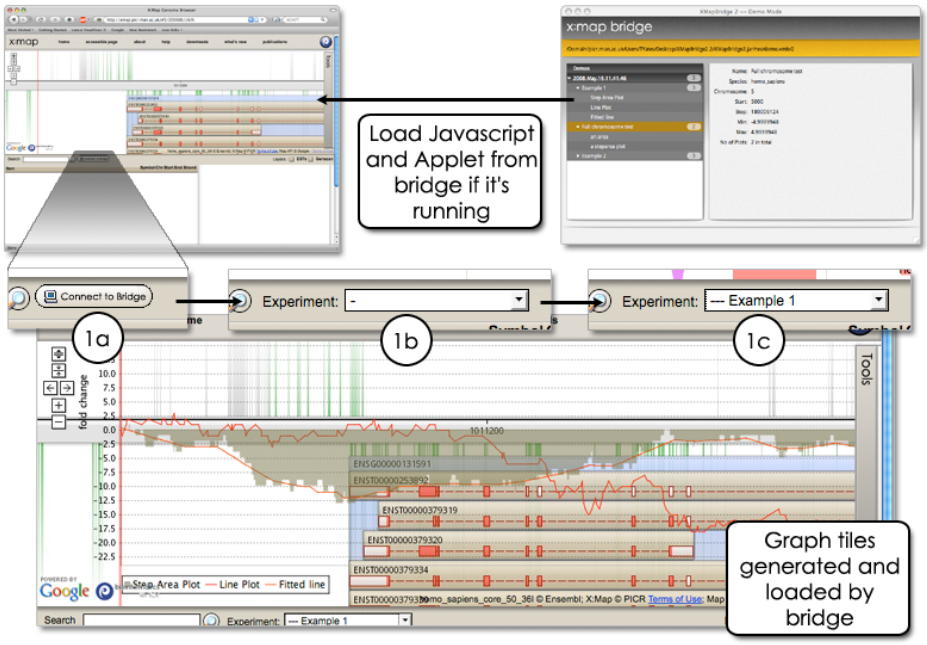

Figure 2: The process to show the graphs in the browser (1a) Click on the connect button (1b) This changes into a drop-down select box (1c) select the graph of choice

If you don't want to use this default directory, then you need to create an environment variable *XMAP_BRIDGE_CACHE* which contains a path to the folder you want to use as a cache folder instead. The details of how to set this variable on multiple operating systems can be found in the *README.TXT* file which was included in the *XMapBridge* download.

The function `xmap.plot` generates a project folder structure, adds a graph to this structure, and adds the plot data in the appropriate directory for *XMapBridge* to find it. So, for example:

```
> library( xmapbridge )
> set.seed( 100 )
> x <- runif( 100, 1100000,1200000 )
> y <- 10 * sin( 2 * pi * ( x-1000000 ) / 10000 )

> xmap.plot( x, y, species="homo_sapiens", chr="1", type="line" )

xmapbridge:Project( /tmp/RtmpECoSrN/2015.Oct.14.20.58.07_0.493.xmb )
```

Will first prompt you to ask if you want to create the default directory (if it's not already there) and then generate a line plot between position 1,100,000bp, and 1,200,000bp

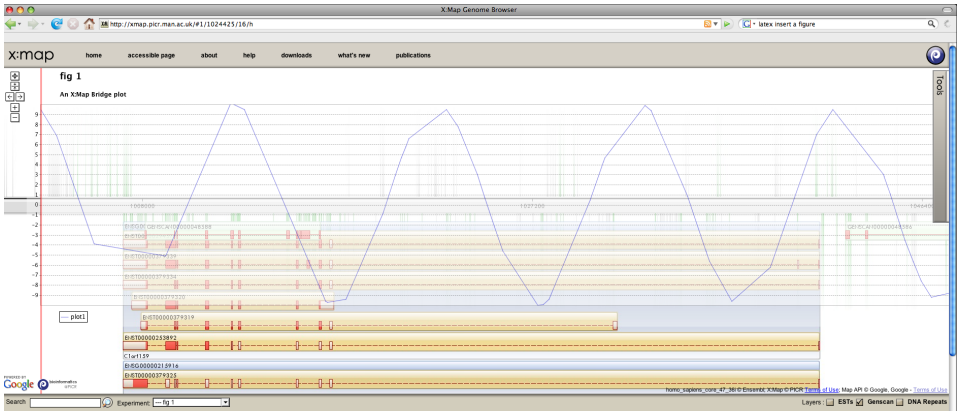

Figure 3: X:Map displaying a simple graph

on human chromosome 1. As you can see from the output, it actually starts at location 1,103,015bp due to the `runif`. (In the following section more complex examples show how to generate graphs for real (Affymetrix exon) array data).

If you re-start *XMapBridge* (assuming it was in demo-mode before), it should now find this directory and the browser (once refreshed or pointed at the X:Map website) should offer you your new graph via the drop-down menu. When you then select it, you should see it displayed, as in figure 3. The next time you use the package, this directory will already exist, so you won't get prompted again, unless you delete or move the cache directory.

> The XMapBridge java application scans your cache folder every few seconds to see if anything has been updated, however these changes can take some time to "bubble" up to the javascript in the browser. There should be no need to refresh the X:Map page and reload the bridge connection applet, but it may take up to 30 seconds for the list of graphs to be updated. When the list is updated, the drop-down selection box reverts back to its initial position, removing any currently viewed graph.

3 Plotting Graphs, the simple way

As we saw above, the simplest way to plot a graph is simply to call `xmap.plot`. As well as x,y coordinates, you also need to specify the chromosome and species. Labels can be provided and the type of plot can also be specified. Later we'll see how to set transparency and colours for the plots.

Here we explore how to generate some plots for Affymetrix Exon arrays. We first load an example dataset (see *?exon.data* for more details on the origins of this dataset):

```
> data(xmapbridge)
```

The dataframe *exon.data* contains information on 1747 probesets targeting 80 different genes, for a triplicate comparison between two cell lines, mcf7 and mcf10a. To generate our plots, we first take our array data and split it into different pieces, one for each gene:

```
> l <- split(exon.data,exon.data$"gene")
```

Then, for each gene we need to extract the appropriate x locations (and which chromosome it is on). Initially, we just do it for the first gene in the list:

```
> g <- l[[1]]
> x   <- g$"seq_region_end" + (g$"seq_region_end" - g$"seq_region_start")/2
> y   <- g[,2]
> chr <- unique(g$"name")
>
```

Now we can plot the data for one of our replicates:

```
> xmap.plot(x,y,chr,species="homo_sapiens",type="area",col="#ff000066")
```

```
xmapbridge:Project( /tmp/RtmpECoSrN/2015.Oct.14.20.58.07_0.493.xmb )
```

Currently, *scatter, line, bar, step, area and steparea* plots are supported. These are be pretty self explanatory - the following should generate plots similar to those in figure 4:

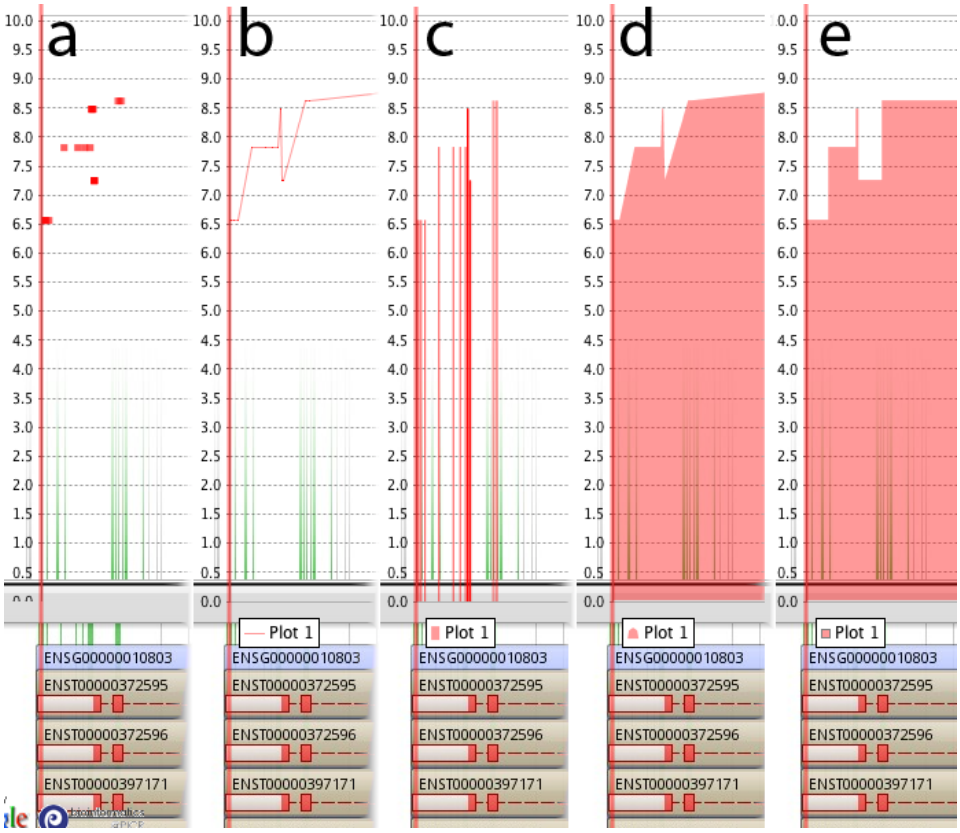

Figure 4: Different styles of plot: (a) scatter, (b) line, (c) bar, (d) area and (e) steparea

```
> xmap.plot(x,y,chr,species="homo_sapiens",type="scatter",col="#ff000066")
```

```
xmapbridge:Project( /tmp/RtmpECoSrN/2015.Oct.14.20.58.07_0.493.xmb )
```

```
> xmap.plot(x,y,chr,species="homo_sapiens",type="line",col="#ff000066")

xmapbridge:Project( /tmp/RtmpECoSrN/2015.Oct.14.20.58.07_0.493.xmb )

> xmap.plot(x,y,chr,species="homo_sapiens",type="bar",col="#ff000066")

xmapbridge:Project( /tmp/RtmpECoSrN/2015.Oct.14.20.58.07_0.493.xmb )

> xmap.plot(x,y,chr,species="homo_sapiens",type="area",col="#ff000066")

xmapbridge:Project( /tmp/RtmpECoSrN/2015.Oct.14.20.58.07_0.493.xmb )

> xmap.plot(x,y,chr,species="homo_sapiens",type="steparea",col="#ff000066")

xmapbridge:Project( /tmp/RtmpECoSrN/2015.Oct.14.20.58.07_0.493.xmb )
```

4 Multiple plots on a graph

XMapBridge can generate a graph with multiple plots on it. This is done by calling `xmap.points` after the initial `xmap.plot` call. Each call to `xmap.points` will add a new *Plot* to the current *Graph*:

```
> #Pick an interesting gene
> g <- l[["ENSG00000128394"]]
> x   <- g$"seq_region_end" + (g$"seq_region_end" - g$"seq_region_start")/2
> chr <- unique(g$"name")
> y <- g[,2]

> xmap.plot(x,y,chr,species="homo_sapiens",type="area",col="#ff000033",ylim=c(0,16))

xmapbridge:Project( /tmp/RtmpECoSrN/2015.Oct.14.20.58.07_0.493.xmb )

> xmap.points(x,y,chr,type="line",col="#ff0000ff")

xmapbridge:Project( /tmp/RtmpECoSrN/2015.Oct.14.20.58.07_0.493.xmb )

> #We were plotting the 1st array (2nd column in g), now plot the 4th (5th column in
> y <- g[,5]

> xmap.points(x,y,chr,type="area",col="#0000ff33")

xmapbridge:Project( /tmp/RtmpECoSrN/2015.Oct.14.20.58.07_0.493.xmb )

> xmap.points(x,y,chr,type="line",col="#0000ffff")

xmapbridge:Project( /tmp/RtmpECoSrN/2015.Oct.14.20.58.07_0.493.xmb )
```

This should look like figure 6. (For each array we first generate the shaded area, and then add the edge as an additional line plot).

> Note: As you can see from the output, these calls to `xmap.points` returns a new *Plot* object that resides within the same *Graph* and *Project* at the plot returned from the initial `xmap.plot` call.

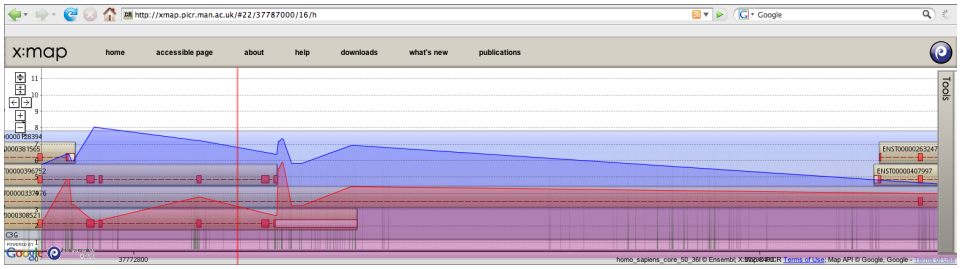

Figure 5: Multiple plots on the same graph

5 More fine grained control

A quick look at the XMapBridge GUI will show that all of the graphs and plots generated so far have been placed in a hierarchy, beginning with a 'Project' and with a series of graphs and plots underneath that.

This is fine if we are happy with all of the plots generated during a single session to be grouped into a single project in this way, but we can express more fine grained control if required.

In the next example, we will create a new project and then iterate along our list of genes, l, generating a new graph for each gene. Each of these graphs will be placed in the new project we've created.

First, we set up some colours and names for our individual plots:

```
> cols.bg <- c(rep("#ff000011",3),rep("#0000ff11",3))
> cols.fg <- c(rep("#ff0000ff",3),rep("#0000ffff",3))
> names   <- c("7.1","7.2","7.3","10a.1","10a.2","10a.3")
```

(We will explain how colours are defined later on).

Then we create a new project and store the resultant id in the variable *pid*:

```
> pid <- xmap.project.new("mcf7 vs mcf10a")
```

Next, we write a function that can plot a single gene. It takes a matrix of data (i.e. one of the elements of l) and the project id:

```
> plot.a.gene <- function(g,pid) {
+   x    <- g$"seq_region_end" + (g$"seq_region_end" - g$"seq_region_start")/2
+   y    <- g[,2:7]
+   chr  <- unique(g$"name")
+   gene <- unique(g$"gene")
+   xlim <- range(x)
+   gph  <- xmap.graph.new(projectid=pid, name=gene,desc="vignette example", min=0, m
+   for(i in 1:6) {
+     xmap.points(graphid=gph,x=x,y=y[,i],type="area",col=cols.bg[i],xlab="")
+     xmap.points(graphid=gph,x=x,y=y[,i],type="line",col=cols.fg[i],xlab=names[i])
+   }
+ }
```

Note how the function `xmap.graph.new` is used to create a new graph. It takes a projectid as a parameter, which specifies which project the graph should be placed in, and returns a graphid. This is used by `xmap.points` to specify which graph to add the plots to.

Finally, we can generate a graph for each of our genes using `lapply` across *l*:

```
> lapply(l,plot.a.gene,pid=pid)
```

6 Specifying Graph Colours

By default, the *XMapBridge* allocates unique colours to each plot in the graph, however, note how in the previous example, the colour of the plot is being set as an RGB value with an extra alpha value specifying the opacity. The format for these colours is *"#RRGGBBAA"* with each of the Red, Green, Blue and Alpha components being a hexadecimal number from 00 to FF.

So, for example, a colour value of *"#00339955"* for the first plot gives a (slightly green) blue colour with the opacity set to 33% so that the underlying images from the X:Map website (and the underlying plots on this same graph) can be seen through the current plot (see figure 6).

```
> library( RColorBrewer )

> #A light grey almost transparent step area
> xmap.plot( x, y, type="steparea", col="#11111111", ylab="intensity",
+            chr="4", species="homo_sapiens" )

xmapbridge:Project( /tmp/RtmpECoSrN/2015.Oct.14.20.58.07_0.493.xmb )

> #With an opaque orange edge
> xmap.points( x, y, type="step", col="#F7941DFF" )

xmapbridge:Project( /tmp/RtmpECoSrN/2015.Oct.14.20.58.07_0.493.xmb )
```

The function `xmap.col` then provides an easy way to set alpha values:

```
> #Five levels of red, chosen by the RColorBrewer
> reds <- brewer.pal( 5, "Reds" )
> reds <- xmap.col( reds, alpha=0x55 )
> for( i in 1:5 ) {
+   xmap.points( x, y / i, type="area", col=reds[i] )
+ }
```

> Note that `xmap.plot` also allows colours to specified as an integer. In this case, the alpha value goes first; the format is *0xAARRGGBB*. `xmap.col`, returns integers in this form.

Figure 6: Specifying colours and alpha values

7 Installation and configuration details

7.1 Installation of the XMapBridge software

The XMapBridge Java application can be downloaded from the X:Map website at `http://xmap.picr.man.ac.uk/download/bridge` as a simple ZIP file. When extracted, there are comprehensive installation instructions to be found in the file *README.TXT*.

7.2 Configuration of the xmapbridge package

xmapbridge and *XMapBridge* can be told to look in another location apart from the default for the directory they share. This is specified by an environment variable with the name *XMAP_BRIDGE_CACHE*, which points to a folder you wish to use. Again, comprehensive instructions for doing this on the main operationg systems can be found in the *README.TXT* file that is part of the *XMapBridge* download.