users who have never encountered them before.

While there are, of course, tasks for which mosaic plot is preferable, we feel that the balloonplot serves admirably to allow high-levels patterns to be quickly perceived by untrained users.

Conclusion

Using the well worn Titanic data, we have shown how balloonplots help to convey important aspects of tabular data, without obscuring the exact numeric values. We hope that this new approach to visualizing tabular data will assist other statisticians in more effectively understanding and presenting tabular data.

We wish to thank *Ramon Alonso-Allende* allende@cnb.uam.es for the discussion on R-help which lead to the development of `balloonplot`, as well as for the code for displaying the row and column sums.

Bibliography

M. Friendly. Graphical methods for categorical data. *Proceedings of SAS SUGI 17 Conference*, 1992.

J. A. Hartigan and B. Kleiner. Mosaics for contingency tables. *Computer Science and Statistics: Proceedings of the 13th Symposium on the Interface. New York: Springer-Verlag*, 1981.

R. D. Snee. Graphical display of two-way contingency tables. *The American Statistician*, 28:9–12, 1974.

Gregory R. Warnes, Pfizer Inc., USA
gregory.r.warnes@pfizer.com
Nitin Jain, Smith Hanley Inc, USA
nitin.jain@pfizer.com

Drawing pedigree diagrams with R and graphviz

by Jing Hua Zhao

Human genetic studies often involve data collected from families and graphical display of them is useful. The wide interest in such data over years has led to many software packages, both commercial and non-commercial. A recent account of these packages is available (Dudbridge et al., 2004), and a very flexible package Madeline (http://eyegene.ophthy.med.umich.edu/madeline/index.html) is now released under the GNU General Public License. A comprehensive list of many packages, including the package LINKAGE (Terwilliger and Ott, 1994) for human parametric linkage analysis and GAS (Genetic Analysis System, http://users.ox.ac.uk/~ayoung/gas.html) for some other analyses, can be seen at the linkage server at Rockefeller University (http://linkage.rockefeller.edu).

Here I describe two functions in R that are able to draw pedigree diagrams; the first being `plot.pedigree` in `kinship` developed for S-PLUS by Terry Therneau and Beth Atkinson and ported to R by the author, and the second `pedtodot` in `gap` based on David Duffy's `gawk` script (http://www2.qimr.edu.au/davidD/Course/pedtodot) that requires `graphviz` (http://www.graphviz.org). Both are easy to use and can draw many pedigree diagrams quickly to a single file, therefore can serve as alternatives to some programs that only offer interactive use.

Representation of pedigrees

The key elements to store pedigrees using a database is via the so-called family trios each containing individual's, father's and mother's IDs. Founders, namely individuals whose parents are not in the pedigree, are set to be zero or missing. Individual's gender (e.g. 1=male, 2=female) is included as auxiliary information, together with pedigree ID in order to maintain multiple pedigrees in a single database, each record of which indicates a node in the pedigree graph.

For instance, information for pedigree numbered 10081 in genetic analysis workshop 14 (GAW14, http://www.gaworkshop.org) is shown as follows.

```
pid id father mother sex affected
10081  1  2  3   2 2
10081  2  0  0   1 1
10081  3  0  0   2 2
10081  4  2  3   2 2
10081  5  2  3   2 1
10081  6  2  3   1 1
10081  7  2  3   2 1
10081  8  0  0   1 1
10081  9  8  4   1 1
10081 10  0  0   2 1
10081 11  2 10   2 1
10081 12  2 10   2 2
```

```
10081 13  0  0   1 1
10081 14 13 11   1 1
10081 15  0  0   1 1
10081 16 15 12   2 1
```

Here all IDs are integers with obvious meanings just described, and the variable `affected` indicates if an individual is alcoholic (1=nonalcoholic, 2=alcoholic) according to DSMIIIR and Feighner definition ALDX1 in the dataset.

In human genetic linkage studies, this is also called pre-makeped format since these IDs can also be string variables, e.g. individuals' names, and a utility program makeped in LINKAGE can be used to generate the serial integer IDs and perform simple checks on errors in family structure(s).

Suppose this is kept in a text file called 10081.pre, we use

```
pre <- read.table("10081.pre",header=TRUE)
```

to read it into object `pre`.

The pedigree-drawing algorithm

Typically, in a pedigree diagram males and females are shown in squares and circles, respectively. Spouses can form marriage nodes from which nodes for children are derived. It is also customary to draw pedigree diagrams top down, so that children at a given generation could have children of their own in the next generation. This implies that the conceptually simple algorithm for pedigree drawing would involve sorting members of a pedigree by generation and align members of the same generation horizontally and those at different generations vertically. In other words, the family is drawn as a directed graph with members as nodes and ordered by their generation numbers. The algorithm could be more involved if there are marriage loops in the family, i.e. overlapping generations, or if the pedigree is too large to fit in a single page. More details on the algorithmic aspects of pedigree-drawing (Tores and Barillot, 2001) can be found for interested readers.

Fortunately, there is software publicly available that implements this algorithm. Among these the most notable is graphviz (`http://www.graphviz.org`) consisting of programs `dot`, `dotty`, `neato` and `lneato`. In the following section two implementations of the pedigree-drawing algorithm with or without use of graphviz will be described.

Drawing pedigree diagrams with R and graphviz

via `plot.pedigree` in `kinship`

Package `kinship` was developed for linear mixed and mixed-effects Cox models of family data, but it has function `plot.pedigree` for drawing pedigree diagrams. As an S3 function for class 'pedigree', it can be used as `plot` if supplied with argument with class 'pedigree'.

```
library(kinship)
attach(pre)
par(xpd=TRUE)
ped <- pedigree(id,father,mother,sex,affected)
plot(ped)
```

This gives Figure 1. We can use R devices such as postscript to keep the diagram as an outside file, e.g., `postscript("10081.ps"); plot(ped); dev.off()`. Either postscript or pdf format can hold more than one pedigree diagrams.

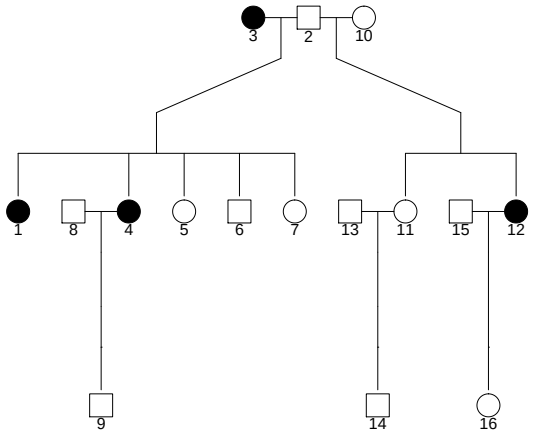

Figure 1: Pedigree 10081 by kinship

via `pedtodot` in `gap`

The diagram produced by `plot.pedigree` in `kinship` is a still image so that nodes in the pedigree graph can not be pulled and dragged. This might not be trivial to implement. However, we can use graphviz to achieve this. A nice feature of the package its ability to interpret dot language, which is in ASCII format that allows for text-editing. Keeping this in mind, we can simply generate a dot file for given pedigree(s) using dot syntax.

```
library(gap)
pedtodot(pre)
```

By default, this generates `10081.dot` which can be used by dot, e.g. dot -Tps -ofigure2.ps 10081.dot giving Figure 2.

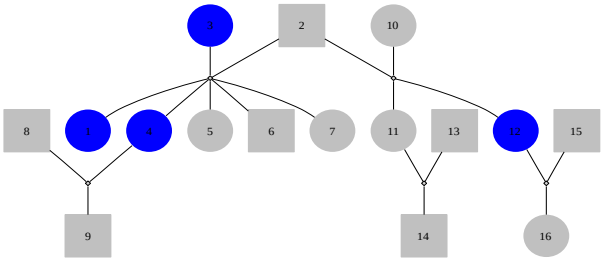

Figure 2: Pedigree 10081 by dot

Alternatively, we use

```
library(gap)
pedtodot(pre,dir="forward")
```

Besides `dot`, we can also use neato -Tps -o figure3.ps 10081.dot, giving a more liberal graph (Figure 3). Its unusual looking makes it appealing for exposing the peeling algorithm in the likelihood calculation of pedigrees.

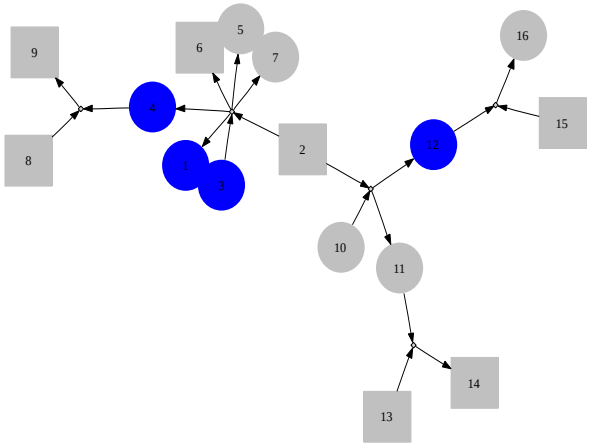

Figure 3: Pedigree 10081 by neato

A single file containing all pedigree diagrams can be generated through the use of R `sink` command and screen output using `sink=FALSE` option in `pedtodot`. For example, given that all the pre-makeped pedigrees are contained in the object gaw14ped, we can use the following command,

```
sink("gaw14.dot")
pedtodot(gaw14ped,sink=FALSE)
sink()
```

to generate a complete list of pedigree diagrams available from the web site http://www.mrc-epid.cam.ac.uk/~jinghua.zhao/r-progs.htm.

Summary and further remarks

Further information about the two functions is available from the packages themselves. Note that `plot.pedigree` in `kinship` does not require pedigree ID be specified, while `pedtodot` in `gap` does. Unlike `plot.pedigree`, `pedtodot` requires `graphviz` to visualize graphics, but can be edited with `dotty`, and be printed out in multiple pages when a pedigree diagram is too big to fit in a single page. Both can produce a set of pedigree diagrams in a single file.

Although `makeped` is a utility program in LINKAGE, this function is also available in package `gap`. If the pedigree file is in `post-makeped` format, then the option `makeped=TRUE` can be used. However, `pedtodot` can also use string IDs, or file in the so-called GAS format in which gender can take values 'm', 'f', etc.

Most pedigrees in current studies have moderate sizes, therefore sparse two-dimensional arrays are used to keep track of marriages and children from them, which are in turn used in generating dot script. This is less efficient than the original `gawk` script in that the latter use individual IDs directly to index the marriage arrays, leading to shorter code and better use of memory. In addition, further improvement is possible by introducing other features available in the dot language.

It is notable that R package `sem` has function `path.diagram` to generate `dot` file and the Bioconductor (http://www.bioconductor.org) package `Rgraphviz` also uses `graphviz`. If more such packages appear, it is desirable to be familiar with the `dot` language and/or have support for it in R. Although as yet human genetic linkage and genome-wide association analysis is not widely conducted with R, this might change in the near future, as has been demonstrated by the great success of the Bioconductor project. I believe the two R functions described in this note will be very useful to researchers in their genetic data analysis.

Bibliography

F. Dudbridge, T. Carver, and G. Williams. Pelican: pedigree editor for linkage computer analysis. *Bioinformatics*, 20(14):2327–2328, 2004.

F. Tores and E. Barillot. The art of pedigree drawing: algorithmic aspects. *Bioinformatics*, 17(2):174–179, 2001.

J. Terwilliger and J. Ott. *Handbook of Human Genetic Linkage*. The Johns Hopkins University Press, Baltimore, 2001.

Jing Hua Zhao
MRC Epidemiology Unit
jinghua.zhao@mrc-epid.ac.uk

Non-Standard Fonts in PostScript and PDF Graphics

by Paul Murrell and Brian Ripley

Introduction

By default, all of the text produced by R graphics devices uses a sans-serif font *family* (e.g., Helvetica), and in a PostScript or PDF plot only text in Western European languages has been handled until recently. This article describes several ways in which the range of font families has been extended for R 2.3.0.

It has been possible to change the overall font family that is used to draw text within a plot (Murrell, 2004), but the choice of font family for PDF and PostScript devices was largely restricted to the set of Adobe Core 14 fonts—Helvetica (4 faces), Times (4 faces), Courier (4 faces), Symbol, and ZapfDingbats (see Table 1). As from R 2.3.0 much more choice is available.

The examples in this article are written following the conventions needed for a modern Unix-alike operating system such as Linux, but will work with minor changes (for example to the locale names) on most other R platforms including Windows.

A little terminology will be helpful. A *character* is an abstract concept, and a *glyph* is a visual representation of a character. Characters can have more than one representation, for example crossed and uncrossed sevens, and the variants on Greek letters much loved by mathematicians (see `?plotmath`). This matters as East Asian languages may write the same character in different ways.[1]

Minor conveniences

The default font family on a PDF or PostScript device can be controlled using the `family` argument. For example, the expression `pdf(family="Times")` opens a PDF device with the default font family set to Times (a serif font). This is the simplest way to select a font family, if the same font (possibly in different faces, e.g. bold) is to be used for all of the text in a plot.

Prior to R 2.3.0, only a fixed set of font families could be specified via the `family` argument (mostly related to the Adobe Core 14 fonts), but it is now possible to specify any *valid* font family as the default. For example, the expression `pdf(family="mono")` opens a PDF device with the default font family set to be a mono-spaced font (by default Courier on PDF and PostScript devices).

The section on "Font databases" later in this article will clarify what constitutes a "valid" font family.

Changing font family in-line

While it has been possible for some time to be able to change the font family within a plot (e.g., have the title in Times and the remaining labels in Helvetica), in the traditional graphics system, this change could only be made via the `par()` function.

It is now also possible to specify the `family` argument in low-level traditional graphics functions. For example, if the default font family was sans-serif, adding a label to a plot with a serif font used to require code of the form ...

```
> op <- par(family="serif")
> text("A label")
> par(op) # restore the setting
```

... but now the same thing can be achieved by the following code.

```
> text("A label", family="serif")
```

Interationalization

As Martin Maechler likes to say, R has 'for ever' supported ISO Latin 1, the character set of the major[2] Western European languages. Further internationalization features were introduced into R in version 2.1.0 (Ripley, 2005), which allowed users in non-Western-European locales to use variable names and produce output in their native language.

There was good support for graphical devices on Windows, and as from R 2.3.0 on NT-based versions of Windows it is even possible to change to another language and output in that language.

The support for the `X11()` graphics device was as good as the X server could provide, often limited by

[1] http://en.wikipedia.org/wiki/Han_unification

[2] The definition is rather circular, as those are the languages written in Latin-1. Icelandic is the most prominent exception.