Stripless: An Alternative Display for Conditioning Plots

Bert Gunter

2016-09-09

Contents

The greatest value of a picture is when it forces us to notice what we never expected to see.
– John Tukey

Often the most effective way to describe, explore, and summarize a set of numbers – even a large set – is to look at a picture of those numbers.
– Edward Tufte

Introduction and Motivation

A central goal of most data analyses is to determine how one or more response variables relate to a set of covariates. In many such studies, the covariates take on only a few distinct values. In others, "shingling" continuous covarates (Cleveland 1993, in which it is referred to as *slicing*) yields a similar effect. Factorial designed experiments in which the design factors may occur at only 2 or 3 levels apiece in order to keep the experimental size manageable, are a common example of this kind of setup.

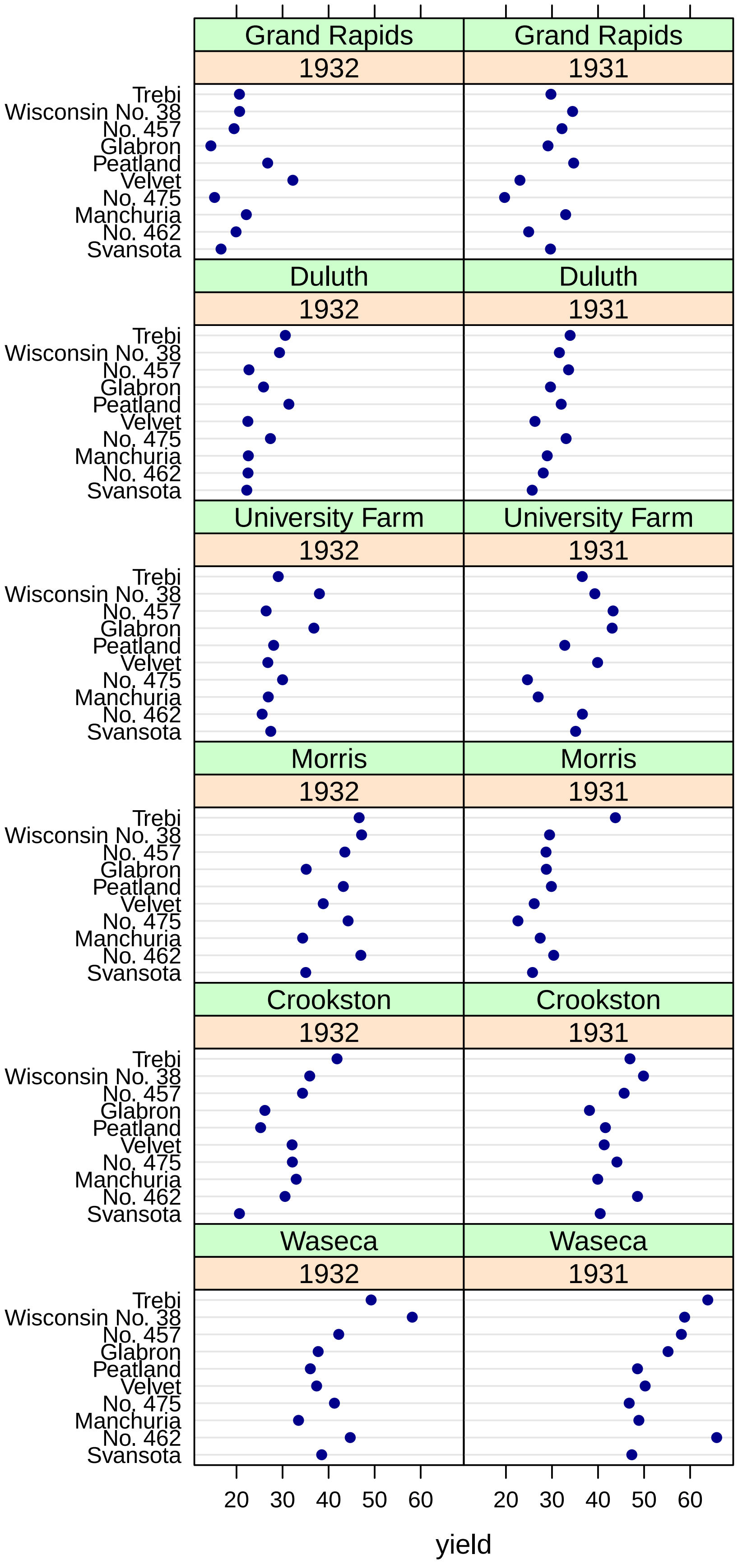
Effective visualization is an essential part of the data analysis process. Searching for and interpreting both expected and anomalous patterns, checking for unusual data that may signal possible problems, and clearly communicating the results are all facilitated by good graphics. Conditional plots (Cleveland 1993), *coplots*, an example of what Tufte called "small multiples" (Tufte 1983), are widely used in such efforts. Trellis Graphics was developed as an implementation of these ideas by William Cleveland, Richard Becker, and colleagues at AT&T Bell Labs and became part the S language system. Subsequently, Deepayan Sarkar developed the lattice package (Sarkar 2008) for R (R Core Team 2015), as an open source implementation. Other R packages, e.g. ggplot2, provide similar capabilities that differ somewhat in the details of how the plots are arranged and annotated. Stripless adds some additional features to lattice, but the same features could very likely be added to others.

I illustrate the concepts with the famous barley data that Cleveland extensively studied in his book and which is available as one of R's stock data sets in the datasets package. The barley data frame has 4 columns, named "yield","variety", "year", and "site" giving barley yields from 10 varieties of barley grown at 6 sites in Minnesota in 1931 and 32 – 120 yields (bushels/acre) in all. The variety, site, and year variables were coded as factors. However, rather than using the default alphanumeric order for the factor levels, they were reordered by their median yields. Thus, for example, since the median of all sites and varieties in 1931 exceeded that of 1932, 1932 is the first level of "year" and 1931 is the second. The following R code does this reordering:

```r
barley[,-1] <- lapply(barley[,-1],function(x)reorder(x,barley$yield,median))
```

As will be clear in a moment for readers not already familiar with Cleveland's book, arranging plots in such informative orderings can reveal important relationships that would otherwise be practically impossible to see. Using this ordering scheme, here is lattice's dotplot of the results (but using "as.table = TRUE" so that median yields increase from top to bottom; Cleveland's display used the default bottom to top ordering).

```r
dotplot(variety~yield|year*site,data=barley,
        col = "darkblue",
        scales = list(alter = 1),
        as.table = TRUE)
```

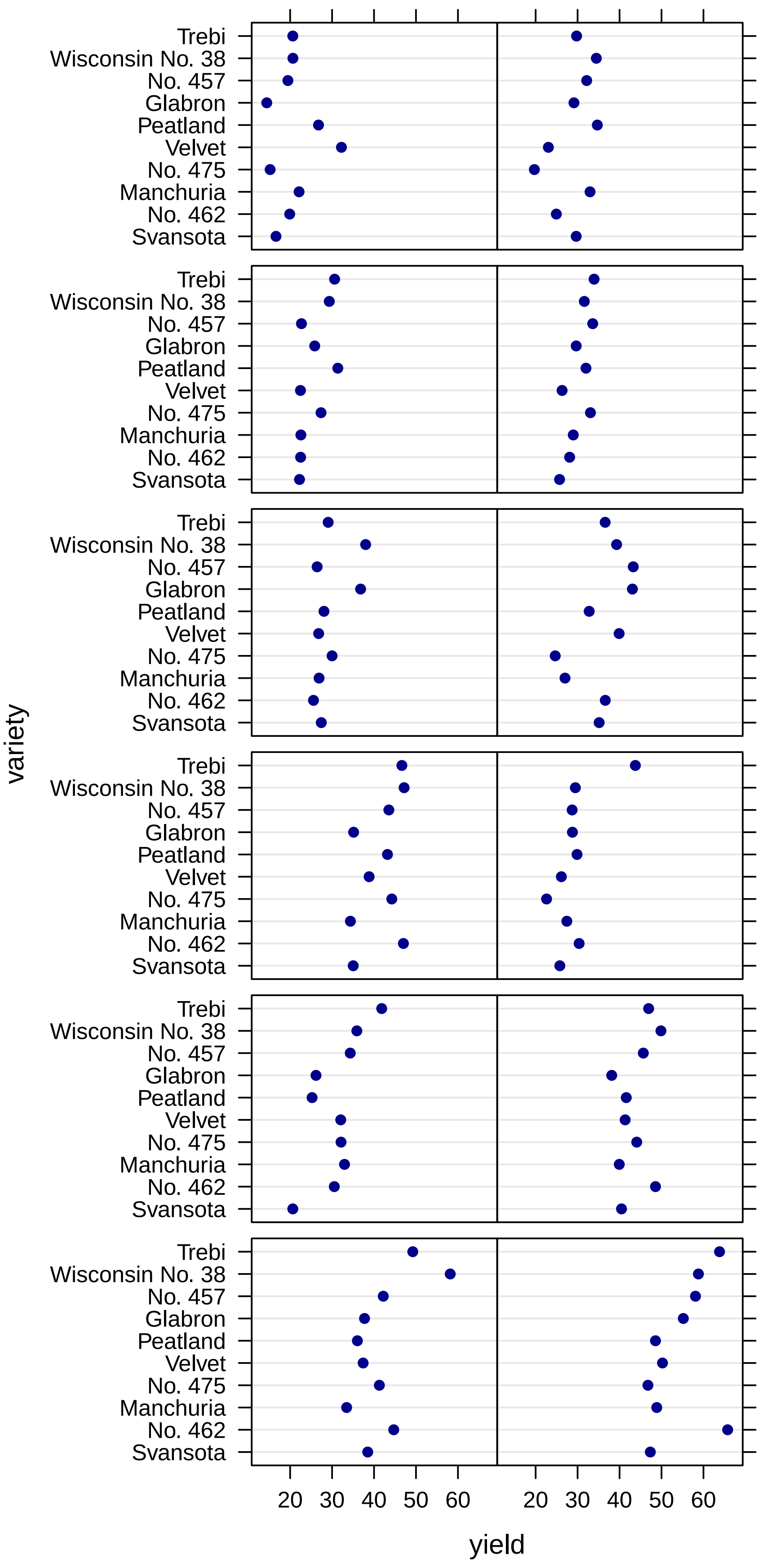

Grand Rapids
1932
1931
Duluth
University Farm
Morris
Crookston
Waseca
Trebi
Wisconsin No. 38
No. 457
Glabron
Peatland
Velvet
No. 475
Manchuria
No. 462
Svansota
20
30
40
50
60
yield

Cleveland's concern was "the Morris anomaly," which reverses the patterns of 1931 yields being higher at all locations and interrupts the parallel increasing trends of location effects for each year. Cleveland makes a convincing case through graphics and further analysis that the two years' results were mistakenly reversed at Morris.

However, my point here is to focus on the architecture of the display, not its content. In particular, replicated strips for location and year labels take up a lot of room, resulting in a squashed data area. While this could be alleviated by fiddling with various parameters of the display like font size, strip height, and so forth – or just enlarging the overall display – the fact is that the labels really aren't necessary: the regularity of the layout permits a separate legend to convey the panel identities without the labeling excess. This is what the `strucplot()` function of stripless does. Here is the `strucplot` version:

```
strucplot(variety~ yield|year*site,data=barley, horizontal=TRUE,
        panel=panel.dotplot,
        col = "darkblue",
        scales = list(alternat = 1),
        spacings = list(x=0, y=.5))
```

```
##
## PLOT STRUCTURE
##
## _Horizontal_
## year:  1932  1931
##
## _Vertical_
## site:  Grand Rapids  Duluth  University Farm  Morris  Crookston  Waseca
```

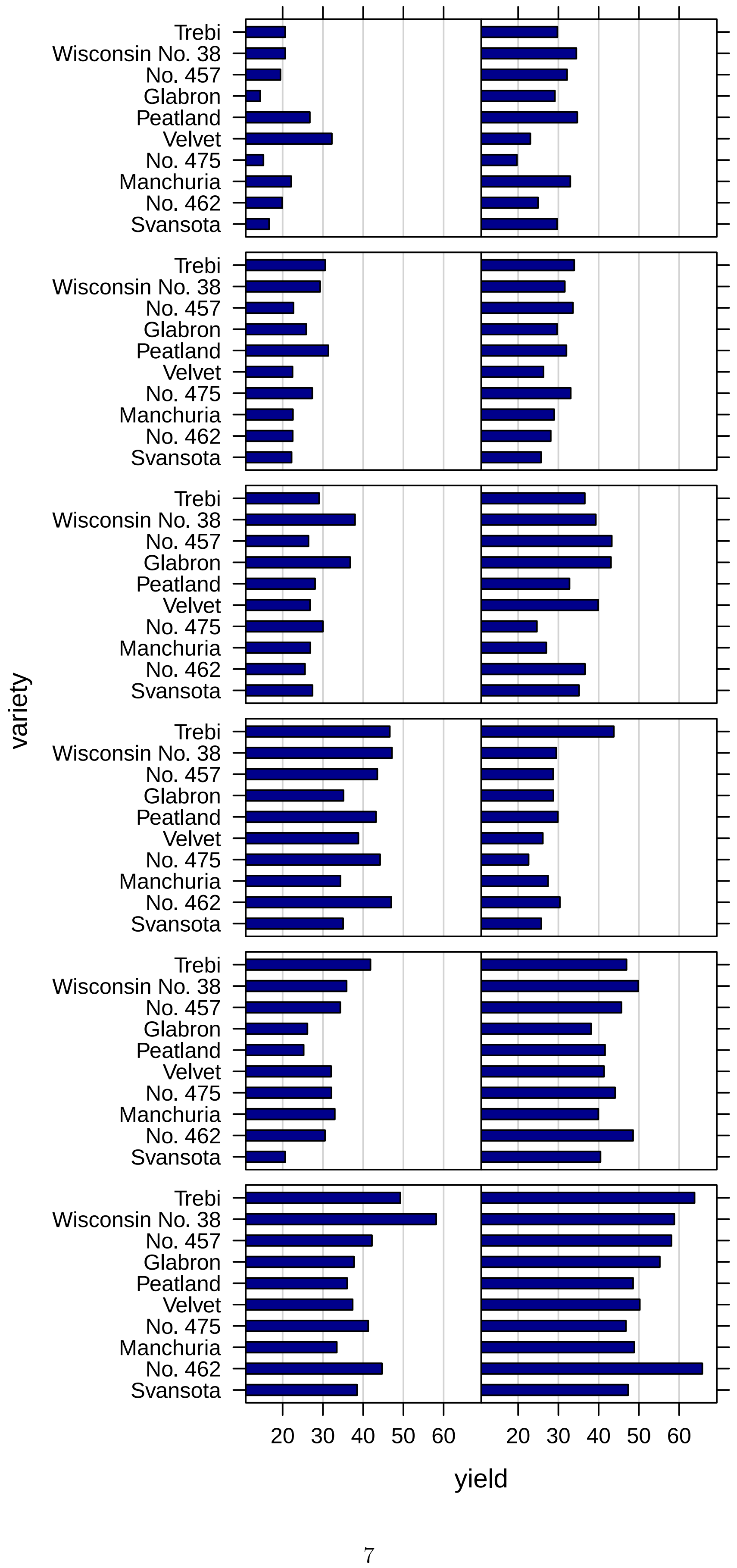

variety
yield
Trebi
Wisconsin No. 38
No. 457
Glabron
Peatland
Velvet
No. 475
Manchuria
No. 462
Svansota
20
30
40
50
60

Note that a legend giving the plot structure is printed on the console. Additionally, there is a `print.stuctured()` method that can be explicitly invoked to add it to a side of the plot or on a separate plot page in addition to the console printout. The "Horizontal" list gives the horizontal factors (only one here) and their levels in order from left to right. For more than one factor, the first varies fastest, the second next fastest, and so forth, as will be described more fully in what follows. The "Vertical" list does the same for the vertical factors (again, only one here) from top to bottom. I think most would agree that strip identifying labels aren't really necessary with this legend.

I think sometimes a bar chart version of the dotplot makes it easier to discern patterns, even though Tufte (1983) considers the bars "chartjunk" because only the ends of the bars convey the information. The stripless panel function `panel.bars()`, which is just a thin wrapper for the lattice `panel.barchart()` function with a few added defaults, provides a quick way to do this.

```
strucplot(variety~yield|year*site,data=barley, horizontal = TRUE,
          panel=panel.bars,
          scales = list(alter = 1),
          spacings = list(x=0, y = .5))
```

```
##
## PLOT STRUCTURE
##
## _Horizontal_
## year:  1932  1931
##
## _Vertical_
## site:  Grand Rapids  Duluth  University Farm  Morris  Crookston  Waseca
```



variety
Trebi
Wisconsin No. 38
No. 457
Glabron
Peatland
Velvet
No. 475
Manchuria
No. 462
Svansota
20 30 40 50 60
yield

By removing the strips and putting their information in the legend, the displays are more compact and, I think, somewhat clearer. Of course, this can only be done when the displays form visual table, i.e. a regular array where the marginals of the array comprise the levels of each factor. But when this is the case, the identifying information in the strips becomes redundant and somewhat 'chart-junky,' especially when there are more than just one or two factors.

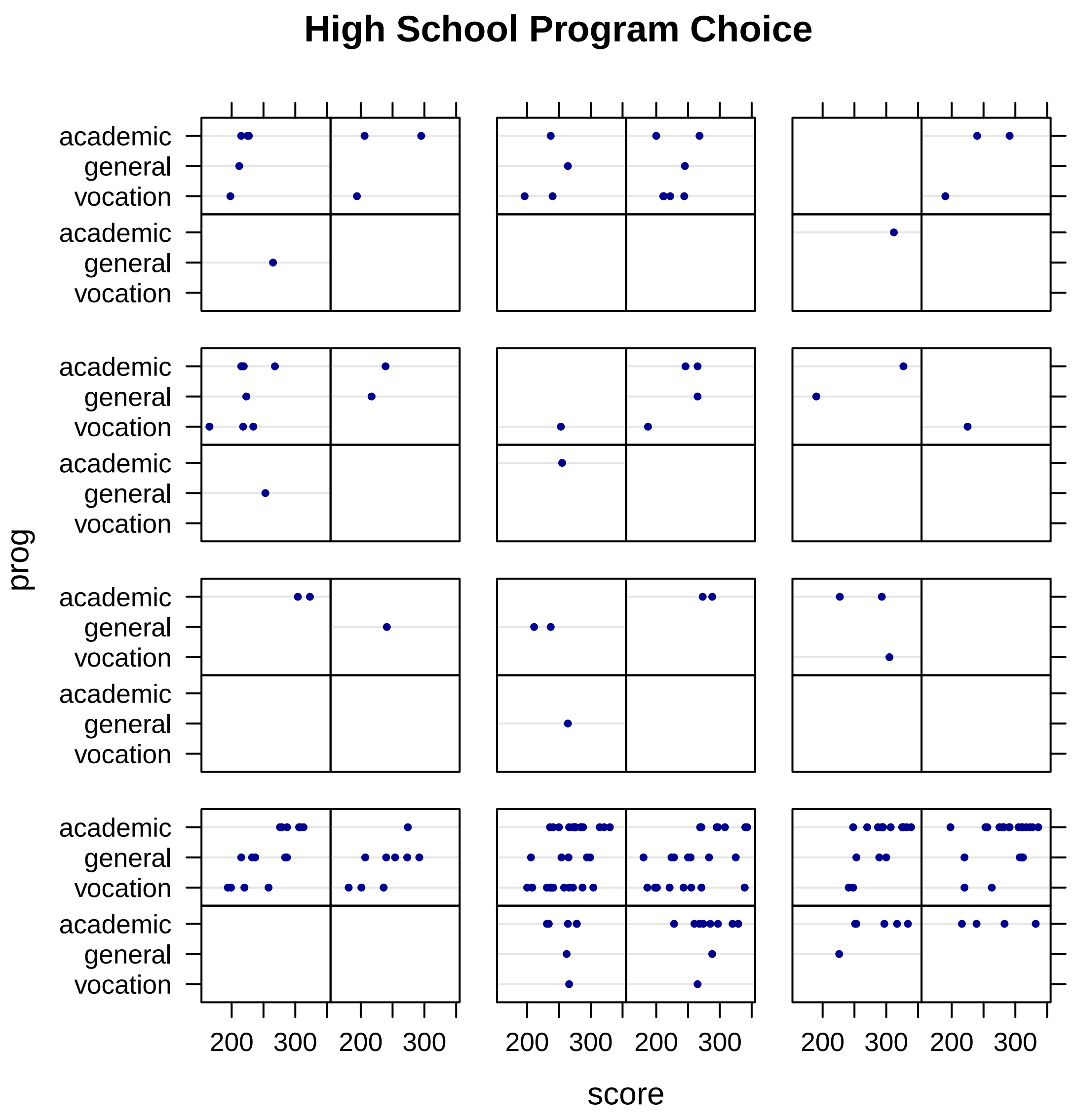
With only three conditioning factors, one of which provides the categories for the dotplot panels, the simple 6 x 2 layout works with or without the panel strips. Here is an example illustrating the difficulties with more factors using the hsb data in the faraway package, which you should first install from your favorite CRAN package repository (there are many interesting data sets in this package). It consists of observations of classification variables (gender, race, public or private school, etc.) and scores on a battery of 5 standardized pre-high school tests for 200 high school students. There was interest in determining which factors influenced the high school program – academic, vocational, or general – that students chose. Rather then do a fine-grained analysis, here, I merely sum the 5 scores for each student and use that total instead of the separate scores. As before, reordering the levels of the categorical factors in some sensible way rather than using the default lexicographic ordering is advisable. I use the median total score for all the factors except `ses`, socioeconomic-economic status, which has a natural ordering of "low"<"middle"<"high".

```r
hsb$score <- rowSums(hsb[,7:11])
hsb[,c(2,3,5,6)] <-lapply(hsb[,c(2,3,5,6)],reorder,hsb$score,median)
hsb$ses <- factor(hsb$ses,levels = c("low","middle","high"))
```

I want to display how the total scores vary with chosen program using the other 4 factors, gender, race, ses, and schooltype as conditioning variables. This means that there will be 2x4x3x2 = 48 panels in all. I use an 8 row x 6 column layout, with gender and ses varying horizontally and schtyp and race varying vertically:

```r
hsbprog <- strucplot(prog~score|gender*race*ses*schtyp,data=hsb,
          main = "High School Program Choice",
          panel= panel.dotplot,
          horizontal = TRUE, cex=.5,
          col = "darkblue",
          scales = list(alternat = 1),
          xyLay= list(x=c(1,3), y = c(4,2)))
plot(hsbprog)
```

```
##
## PLOT STRUCTURE
##
## _Horizontal_
## gender:  female  male
## ses:  low  middle  high
##
## _Vertical_
## schtyp:  public  private
## race:  hispanic  african-amer  asian  white
```



Here's how the the legend and spacings of the panels identifies the factors and their levels. The first horizontal factor is gender, so the first column of juxtaposed paired panels (= 0 spaces between) is for "female" and the second is for "male", with **overall** median "male" scores greater than **overall** median "female" scores (which might of course be reversed within any particular pair or row of pairs). The second horizontal factor is ses, so the first pair of columns of juxtaposed panels is for "low" ses, the second is for "middle", and the third is for "high". The same is true vertically: the juxtaposed pairs of rows are for "public" above and "private" below; the four pairs or rows are the four different levels of race, from "hispanic" on top to "white" on bottom. Median **overall** scores for each factor increase **down** the array.

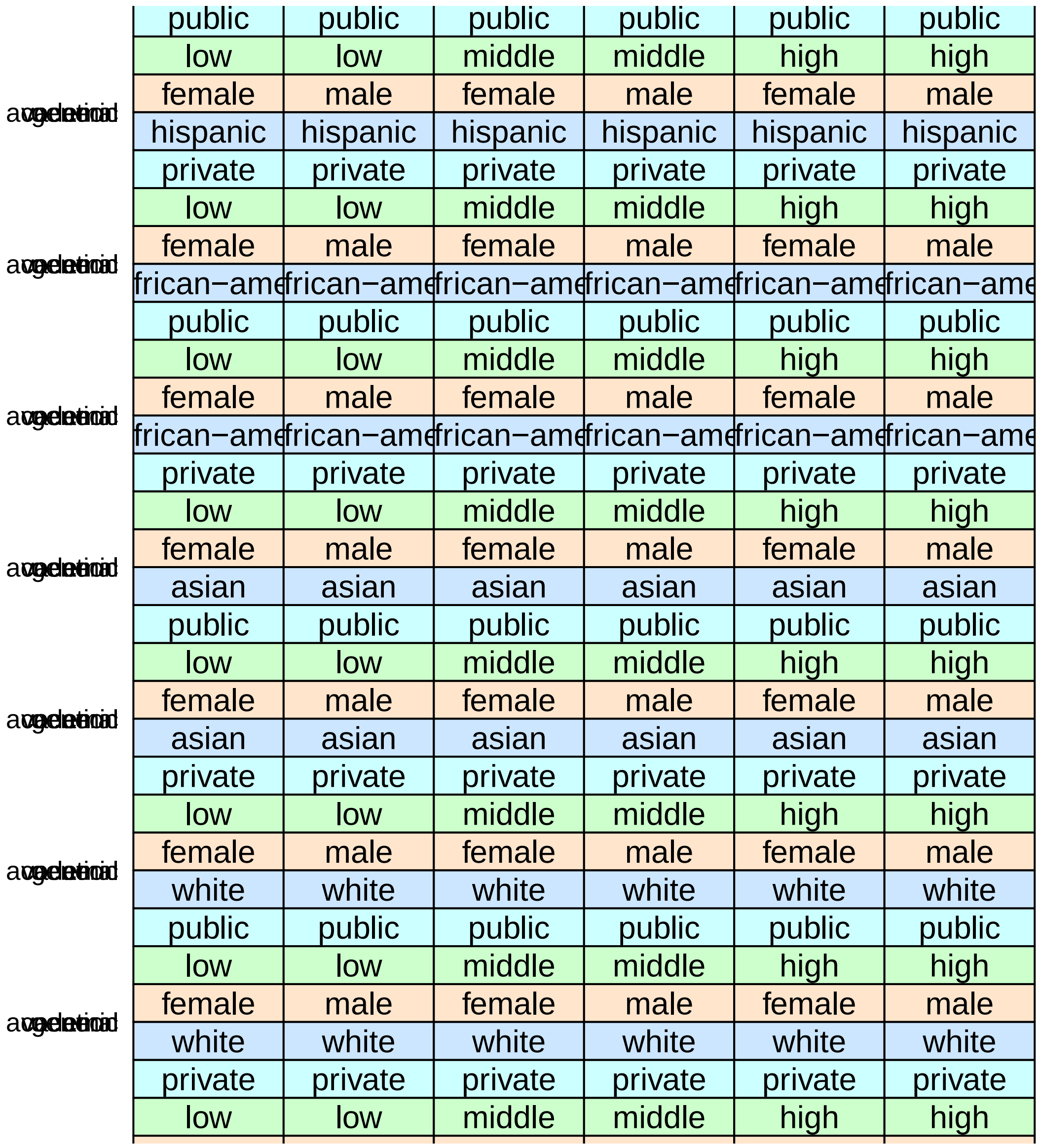
It is thus immediately clear from this plot that:

- Essentially all data are from "public" schools, so including schtype as a covariate is unlikely to be useful
- Almost all the data are for race = "white", so including race as a covariate is unlikely to be useful
- There appears to be a clear association between choice of program and total score on the exams in the way one would expect (highest scores tend to choose "academic"; lowest choose "vocation"; middle choose "general")
- It appears that those in the "high" ses group tend to choose "academic", again as one might expect

One should certainly now consider how to improve this display, perhaps using statistical methods to help refine or modify it. However, I think it stands on its own as a reasonably informative first look at the data.

So the question is: how well would this work with strip labeling included? Here's a default version with the same overall size.

```r
dotplot(prog~score|gender*ses*schtyp*race,data=hsb,
        main = "High School Program Choice",
        layout = c(6,8),
        col = "darkblue",
        scales = list(alter = 1),
        as.table = TRUE)
```



Clearly, this is useless. Of course it could be made to work by, for example, reducing the font size and vertical width in the labels, enlarging the display size, perhaps encoding gender by symbol to reduce the number of factors, and so forth. But having to work so hard to get something useful makes the point: the strip labels have become chartjunk that hide rather than reveal what the data have to say. Removing and replacing them by a simple legend, which is made possible by the regularity of the layout, is a better solution.

Features and Syntax

`strucplot()` is a S3 generic that is a wrapper for the lattice package's `xyplot()` function. The default formula method in the previous examples has basically just 2 new arguments, `spacings` and `xyLayout`(there is also a 3rd, `center` used for 2 level experimental designs that will be discussed later). A few `xyplot` arguments, such as 'layout', 'between', 'skip','strip', 'perm.cond', 'index.cond', and 'drop.unused.levels', are set by `strucplot` and are therefore ignored in a `strucplot` call; but most other `xyplot` arguments, especially the 'panel' function, 'scales', 'groups', and those like 'cex', 'pch', and 'col' that control the plot characteristics, are passed to `xyplot` and so work as expected.

The `formula` argument is as in `xyplot()` (whose Help page should be consulted as needed), except for an added convenience shortcut: a "." on the righthand side after the "|" conditioning symbol means "all other variables in the data.frame `data` argument in their column order." Consequently, it is an error to omit the `data` argument when "." is used. When the conditioning variables are explicitly given, they will be looked up in the usual way.

The `spacings` Argument

`spacings` controls the spacings in the display. It must be a list with "x" and "y" components, each of which must be a nondecreasing sequence of positive numbers. These are used in order to specify the spacing (in character heights) between the levels of the factors in each direction. For example, in the high school students plot, the default spacing of 0:9 means that in the x direction, the levels of gender ("male", "female") are 0 spaces apart (i.e. juxtaposed) and the levels of ses ("low","medium","high"), the pairs of plots in each row, are 1 space apart. Similarly for schtype and race vertically. The default is 0:9 in both directions and usually does not need to be changed. Note that depending on the device, fractional values can also be used.

The `xyLayout` Argument

`xyLayout` controls the layout of the plot – which factors are used for conditioning and in what order – in the x and y directions. In its most general form, it is again a list with "x" and "y" components, each of which must be a vector of indices into the sequence of conditioning variables, i.e. the variables to the right of the "|" in the `formula` argument. The union of these vectors must be the sequence 1,2, ... ,n, where 'n' is the number of conditioning variables; and they must not have any values in common. So in the example, n = 4, and `list(x = 1, y = 2:4)`, `list(x = c(1,3), y = c(4,2))` are both acceptable; while `list(x = 1, y = 3:4)` would not be.

There are also several different ways of abbreviating this specification. The help file for `strucplot` should be consulted for details, but the idea is that anything that allows the full list specification to be "figured out" is acceptable. So, for example, `xyLayout = list(x = 1)` or even `xyLayout = 1` is the same as `list(x = 1, y = 2:4)`. But note that `list(x = c(1,3))` means `list(x = c(1,3), y = c(2,4))` not `list(x = c(1,3), y = c(4,2))`; for the latter, `list( y = c(4,2))` would work.

As indicated above, the x component gives the factors that vary from left to right with levels of the first factor spaced the closest, the second the next closest, etc.. When `as.table` is TRUE, which is the default for `strucplot` (the default for `xyplot` is FALSE), the y component does the same for the y factors from **top to bottom**. If `as.table` is FALSE, then the order is from **bottom to top**.

Print Options

`strucplot()` returns an S3 object that inherits from `class c("structured","trellis")`. There is a `print.structured` method whose default is to print the legend to the console and then call the trellis print/plot method to make the plot, passing to it all arguments given in the call. Consequently, all parameters and options of `print.trellis` are available to any device supported by `trellis.device`.

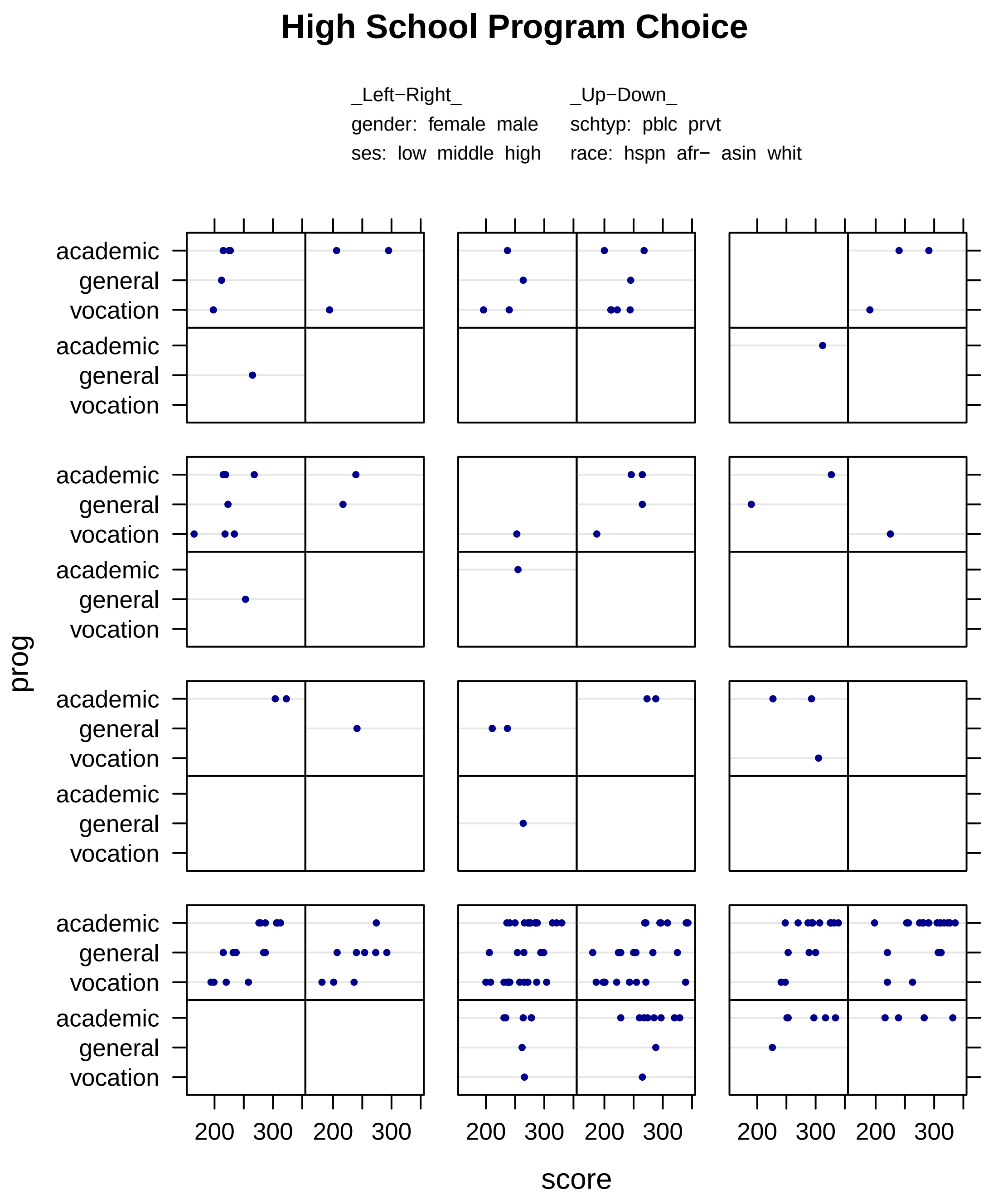
Some limited customization of the console legend can be done. Most important, factor names and/or levels can be shortened to "fit"" using the `abbrevLength` argument, which is used as the `minlength` argument to R's `abbreviate` function. Additional optional arguments can be used to change default legend titles and headings.

However, it may be desirable to show the legend directly on the trellis plot. The optional `legendLoc` and `legend` arguments enable this. The first can be one of the character strings "left", "right", "top", "bottom", or "newpage", and specifies in which margin of the trellis display the legend should be placed. The second must be a **function** that returns a grid "grob", a "graphical object" as described in the `grid` package(Murrell 2011). This function is passed the "structure" attribute of the object to be plotted, the `legendLoc` value, the `abbrevLength` argument, and plots the returned grob in the specified legendLoc. This is done by adding or extending the `legend` argument of the plot object, so that so long as long as there is no conflict with any existing `key` or `legend` component of the `strucplot` call that created it, the new legend will peacefully coexist with any that are already part of the display. A default legend function, `defaultStrucLegend` produces a legend similar to the console's, but appropriately formatted for whichever margin it appears in. It's Help page should be consulted for the full argument list that controls its appearance, some of which it shares with the console legend.

As an example, here is the hsbprog plot with the legend added at the top of the plot with the title omitted, the heading changed, and the race and schtyp levels abbreviated to 4 letters in smaller font. Note that the console legend retains its title, "PLOT STRUCTURE" but applies the other changes.

```r
print (hsbprog,legendLoc = "t",
       cex.font = .7,
       head = c("Left-Right","Up-Down"),
       abbrev = list(race=c(0,4), schtyp = c(0,4)))
```

```
##
## PLOT STRUCTURE
##
## _Left-Right_
## gender:  female  male
## ses:  low  middle  high
##
## _Up-Down_
## schtyp:  pblc  prvt
## race:  hspn  afr-  asin  whit
```



Of course, there is a tradeoff here: If, to minimize the space used, the legend text is abbreviated so much that it is difficult to decipher, it defeats its purpose. This means that one must consider:

- The display format for the plot: how much space overall is available for the display.
- The level of detail that must be clearly shown. Generally, one wants all the information in the plot to be clear, but sometimes a more compact plot in which some detail is lost is appropriate.
- The amount of space needed for a clear and intelligible legend. This may depend on the familiarity that the expected viewer has with the subject matter!

Additional Methods

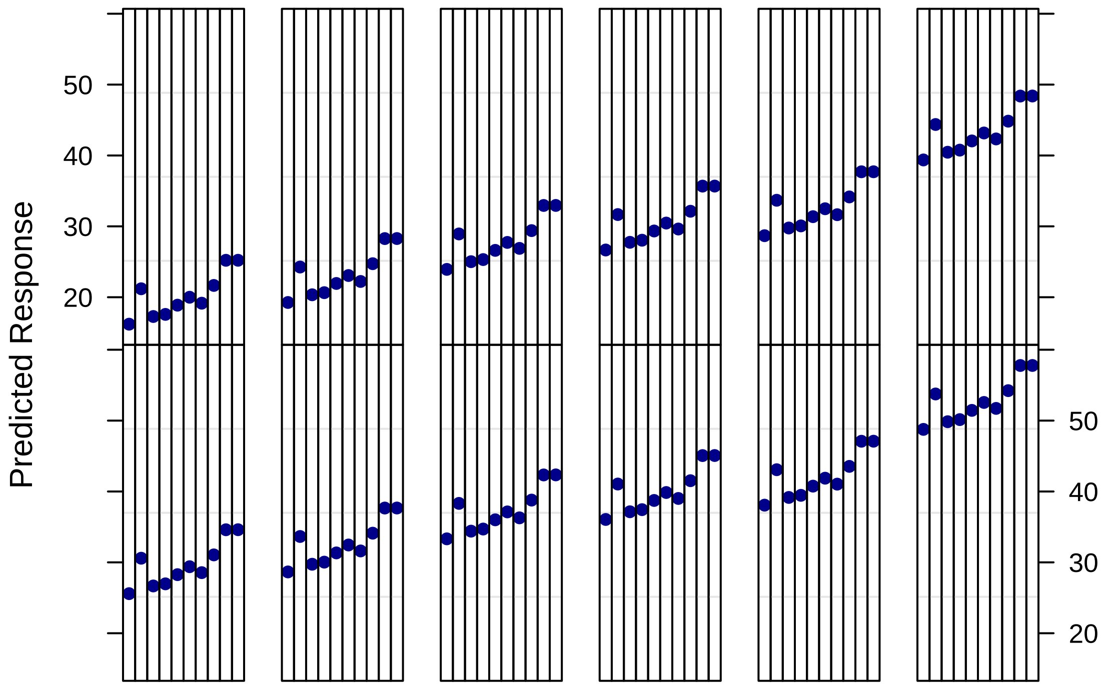
Plotting Results From A Fitted Model

One of the purposes of trellis displays is to make manifest structural relationships between response(s) and covariates. A natural application of this capability is to display predicted values of a fitted model in a trellis plot. `strucplot.lm` does this directly for models inheriting from the "lm" class without having to explicitly call `strucplot` with the predicted values in a formula. In addition to the fitted model object, it can take an optional 'newdata' argument to plot predicted values at data other than those used in the fit. This argument is merely passed down to the relevant predict method. An optional list of additional named arguments, predictArgs, can also be specified if the `predict` method requires them.

For example, here is the result of fitting a simple additive model after first "correcting"

```
barley <- within(barley,{changed_year <- year
    changed_year[site=="Morris" & year == 1932] <- 1931
    changed_year[site=="Morris" & year == 1931] <- 1932}
)
barleyFit <- lm(yield ~ variety + site + changed_year, data=barley)

strucplot(barleyFit, pch=16, col = "darkblue",
         panel= function(...){panel.grid(h=3,v=0); panel.xyplot(...)} )
```

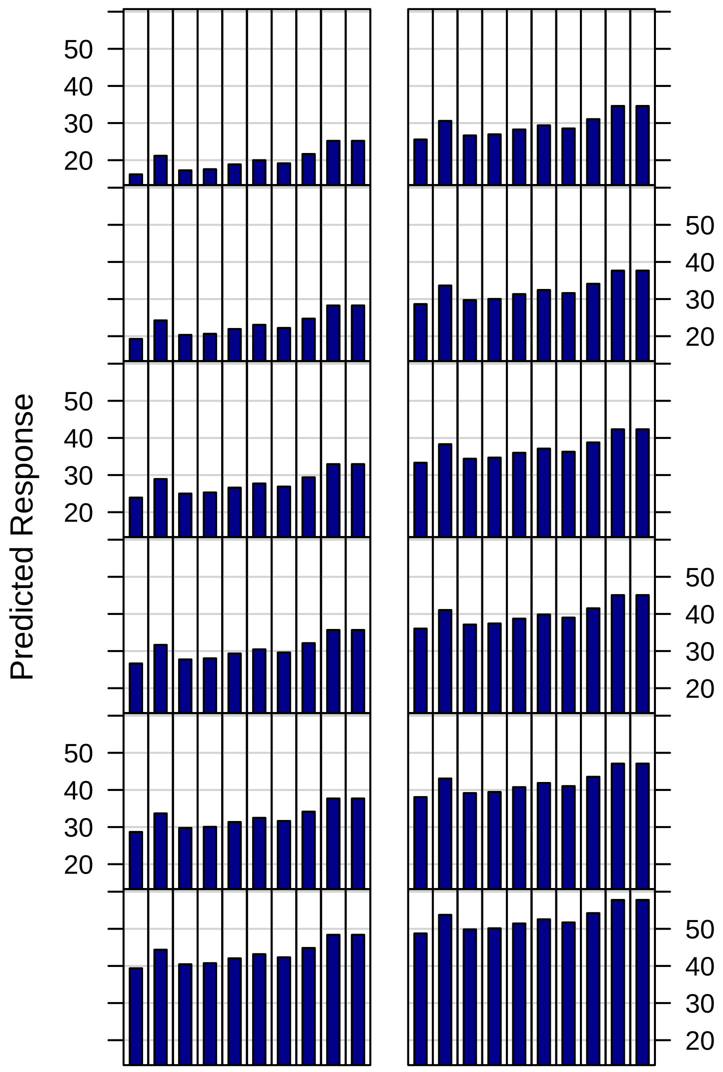
```
##
## PLOT STRUCTURE
##
## _Horizontal_
## variety:  Svansota  No. 462  Manchuria  No. 475  Velvet  Peatland  Glabron  No. 457  Wisconsin No. 38
## site:  Grand Rapids  Duluth  University Farm  Morris  Crookston  Waseca
##
## _Vertical_
## changed_year:  1932  1931
```



Note that the `strucplot` method for a fitted model is of the form `strucplot(~ fitted(modelFit) | v1*v2*...)`, where the conditioning variables after the "|" are the covariates used in the fitted model. As in any `strucplot` call, the plot layout can be changed via the `xyLayout` argument and other panel functions can be used:

```r
strucplot(barleyFit, pch=16, col = "darkblue",
          xyLayout = list(y = 2),
         panel= panel.bars )
```

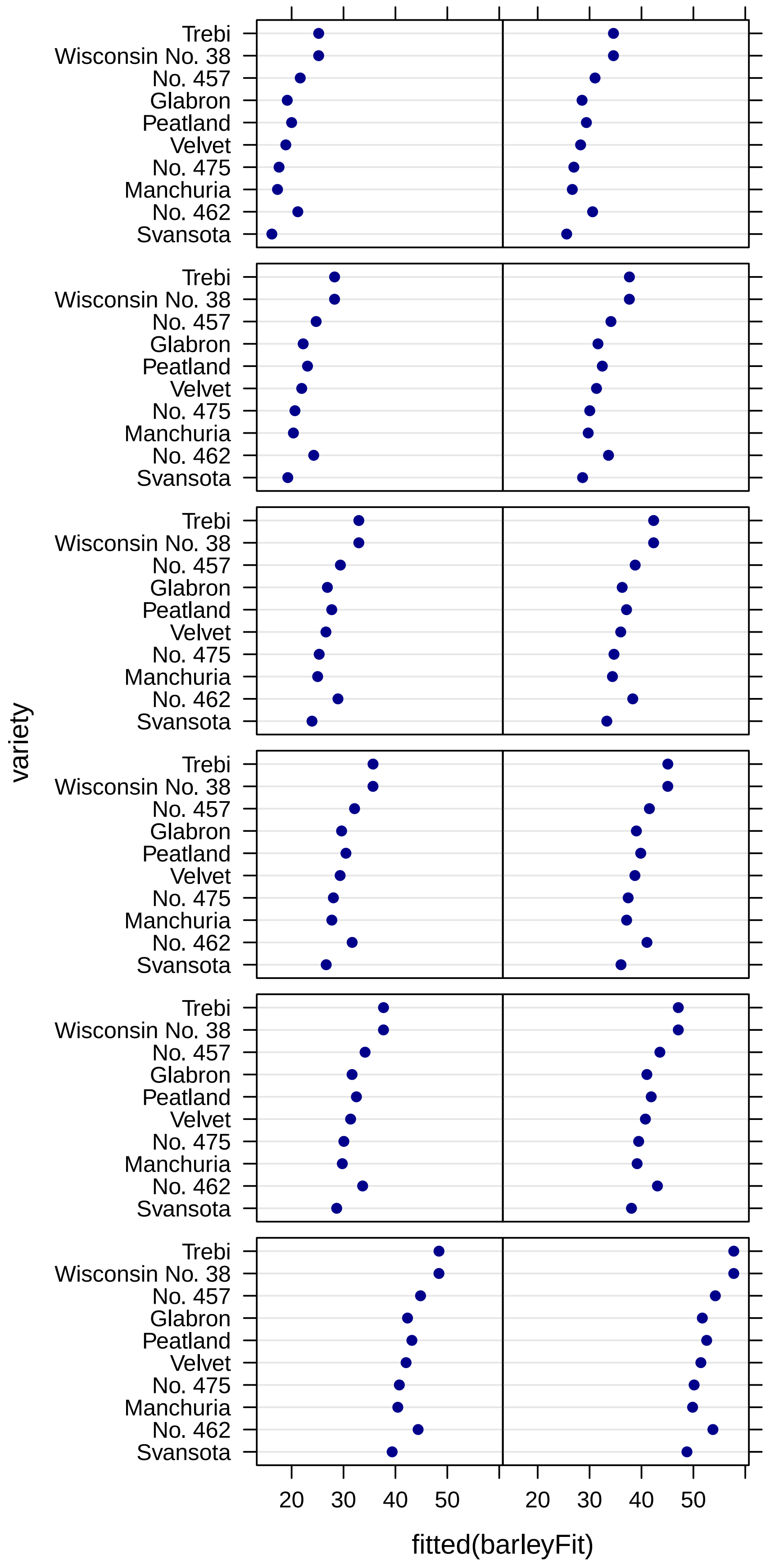
```
##
## PLOT STRUCTURE
##
## _Horizontal_
## variety:  Svansota  No. 462  Manchuria  No. 475  Velvet  Peatland  Glabron  No. 457  Wisconsin No. 38
## changed_year:  1932  1931
##
## _Vertical_
## site:  Grand Rapids  Duluth  University Farm  Morris  Crookston  Waseca
```



However, if a different formula or only a subset of the variables is to be plotted, then the fitted values must be explicitly used in the formula. For example, to reproduce the original display plotting the fitted values in place of the actual values (with the corrected years), one would use:

```r
strucplot(variety~ fitted(barleyFit)|changed_year*site,data=barley, horizontal=TRUE,
        panel=panel.dotplot,
        col = "darkblue",
        scales = list(alternat = 1),
        spacings = list(x=0, y=.5))
```

```
##
## PLOT STRUCTURE
##
## _Horizontal_
## changed_year:  1932  1931
##
## _Vertical_
## site:  Grand Rapids  Duluth  University Farm  Morris  Crookston  Waseca
```



variety
Trebi
Wisconsin No. 38
No. 457
Glabron
Peatland
Velvet
No. 475
Manchuria
No. 462
Svansota
20
30
40
50
fitted(barleyFit)

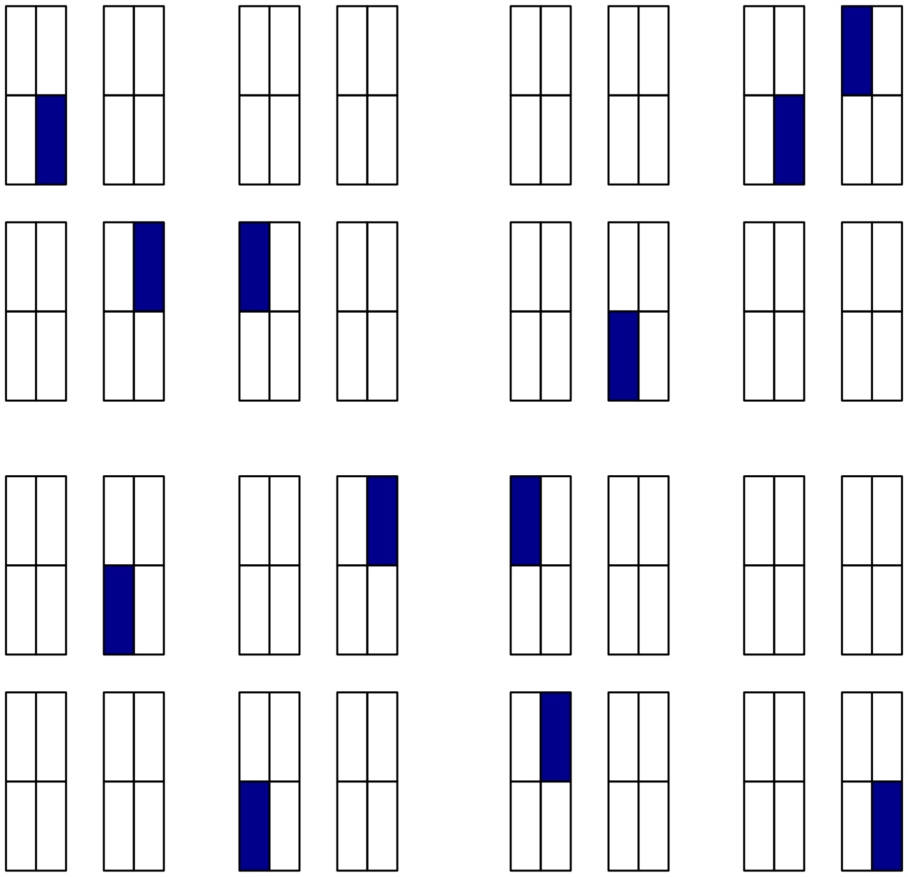
Experimental Design and the `data.frame` Method

One of the motivations for this package was to allow better visualization of factorial designs, especially the 2-level (fractional) factorial designs widely used in industrial and business applications (Box, Hunter, and Hunter 2005). Standard approaches such as marginal plots of data or fits, and so-called *cube plots* (e.g. as in Box, et. al. or the `cubePlot()` function in the R FrF2 package) may limit insight into the full multivariate nature of the designs, because they project onto lower dimensional subspaces – 1 or 2 dimensions for marginals, 3 or 4 for cube plots. When there are more experimental factors, it can be useful to extend the visualization to 5 or more variables. The `data.frame` method does this.

As an illustration, let's compare a a 7 factor, 16 run, 2 level fractional factorial design with a 7 factor, 12 run Plackett-Burman design. To keep this vignette reasonably self-contained, the code below generates the design manually; but there are many R experimental design packages that will do this – and much more, of course! Consult the experimental design task view at https://cran.r-project.org/web/views/ExperimentalDesign.html for details.

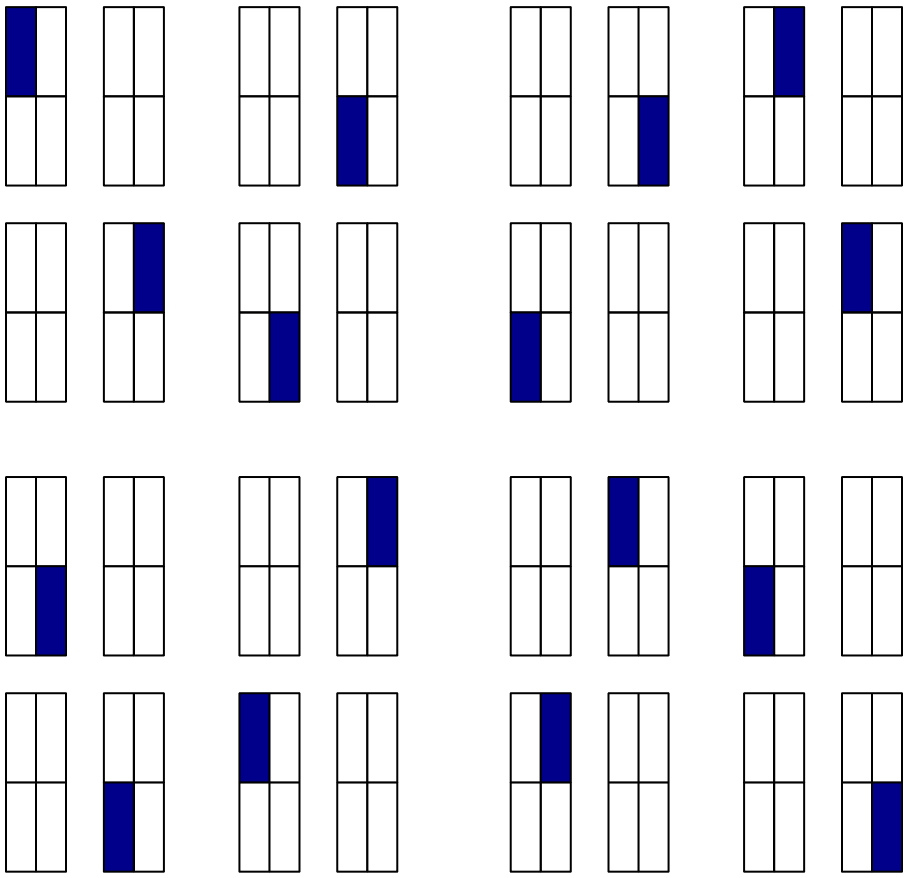
```
## A 7 factor 12 run PB design matrix (using first 7 columns of PB 12 matrix)
pb12 <- matrix(1, nrow=12,ncol=11,dimnames = list(NULL,LETTERS[1:11]))
pb12[1,]<- c(-1,1,-1,rep(1,3),rep(-1,3),1,-1)
for(i in 1:10)pb12[i+1,] <- c(pb12[i,11],pb12[i,-11])
pbex <- data.frame(pb12[,1:7])

## A 7 factor 16 run ff of Resolution IV
ffex<- expand.grid(rep(list(c(-1,1)),4))
names(ffex) <- LETTERS[1:4]
ffex <- within(ffex,{G <- A*B*C; F <- B*C*D; E <- A*B*D})

strucplot(pbex)
```

```
##
## PLOT STRUCTURE
##
## _Horizontal_
## A:  -1  1
## B:  -1  1
## C:  -1  1
## D:  -1  1
##
## _Vertical_
## E:  -1  1
## F:  -1  1
## G:  -1  1
```



```
strucplot(ffex)
```

```
##
## PLOT STRUCTURE
##
## _Horizontal_
## A:  -1  1
## B:  -1  1
## C:  -1  1
## D:  -1  1
##
## _Vertical_
## E:  -1  1
## F:  -1  1
## G:  -1  1
```



As usual, the `xyLayout` argument can be used to change the plot layout.

I think these plots make several useful points.

1. Relationships among even this many or more variables can be fairly easily discerned when displayed as such a regular layout. The human eye/brain pattern recognition capability can be quite effective at extracting meaning given appropriate prompting.
2. Obviously, panel/row/column labels would get in the way.
3. The abundance of empty space makes clear the extreme sparsity of these designs in covering the full multivariate space of possibilities. This is typically lost in lower dimensional projections. I believe it serves as a useful caution in over-interpreting fitted results.
4. There is a clear symmetry in the ff that is absent in the PB. Each of these designs has both pros and cons that are not our concern here; but it is useful to at least show this visually.

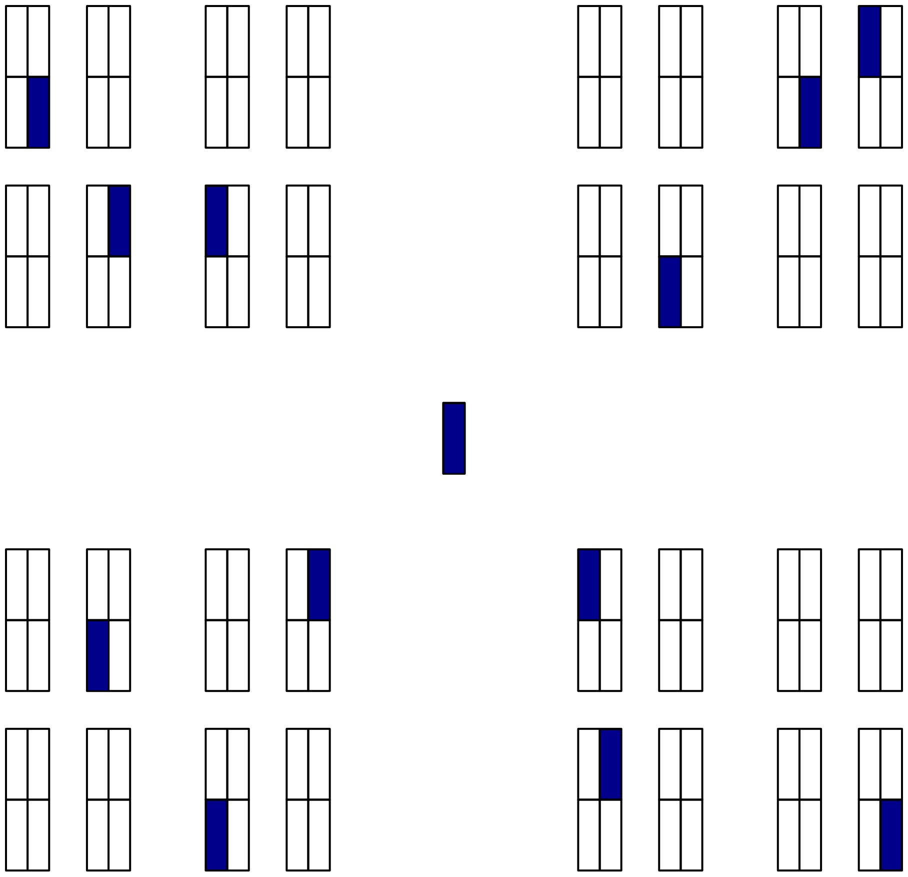
Center points and the `center` argument

Two level designs are geometrically a subset of the 2^n corners of an n-dimensional hypercube, where n is the number of design variables. If all the variables are continuous, there is an awful lot of space in the middle where there is no information, i.e. design points. When this is the case, the designs are sometimes supplemented by adding a (often replicated) point in the "center" in which all the variables are set at a value between their extremes (though not always the midpoint). Plotting such a design as is would usually be a disaster: for instance, the 7 factor example would be a trellis plot with $3^7 = 2187$ separate panels, most of which would be empty, too small to render properly, and would take forever to display.

To deal with this, a more compact display that omits all the empty panels and places the single centerpoint panel in the middle of the plot can be produced by adding the `center = TRUE` argument to the call. As an example, augment the Plackett-Burman design with a center point and set the `center` argument to TRUE.

```
pbex <- rbind(pbex,c(0, rep(c(-.5,.5),3)))
strucplot(pbex, center = TRUE)
```

```
##
## PLOT STRUCTURE
##
## _Horizontal_
## A:   -1  0  1
## B:   -1  -0.5  1
## C:   -1  0.5  1
## D:   -1  -0.5  1
##
## _Vertical_
## E:   -1  0.5  1
## F:   -1  -0.5  1
## G:   -1  0.5  1
```



Note that the "center" point is shown in the exact center of the display, even though the values of the individual factors are mostly not at the midpoints of the extremes.

Examples

Let's now use these plots to analyze some real data. The weldstrength data set in the `farwaway` package is a 9 factor, 16 run ff (a 2^{9-5}) exploring the effect of 9 factors on the strengths of welds. The low and high levels of the factors are coded as 0 and 1. A barplot of the response with all 9 factors using the default xyLayout of 5 factors across the rows and 4 factors down the columns would be produced by the following code, but the display is not shown: there would be 2ˆ9 = 512 separate panels, only a few of which contain data.

```
strucplot(~Strength|., data = weldstrength, panel = panel.bars,
main = "Full Weldstrength Design")
```

```
##
## PLOT STRUCTURE
##
## _Horizontal_
## Rods:  0  1
## Drying:  0  1
## Material:  0  1
## Thickness:  0  1
## Angle:  0  1
##
## _Vertical_
## Opening:  0  1
## Current:  0  1
## Method:  0  1
## Preheating:  0  1
```

A reasonable alternative is to first run a regression to see if the number of factors to display can be reduced to those few that may reveal interesting structure.

```
round(coef(lm(Strength~., data = weldstrength)),2)
```

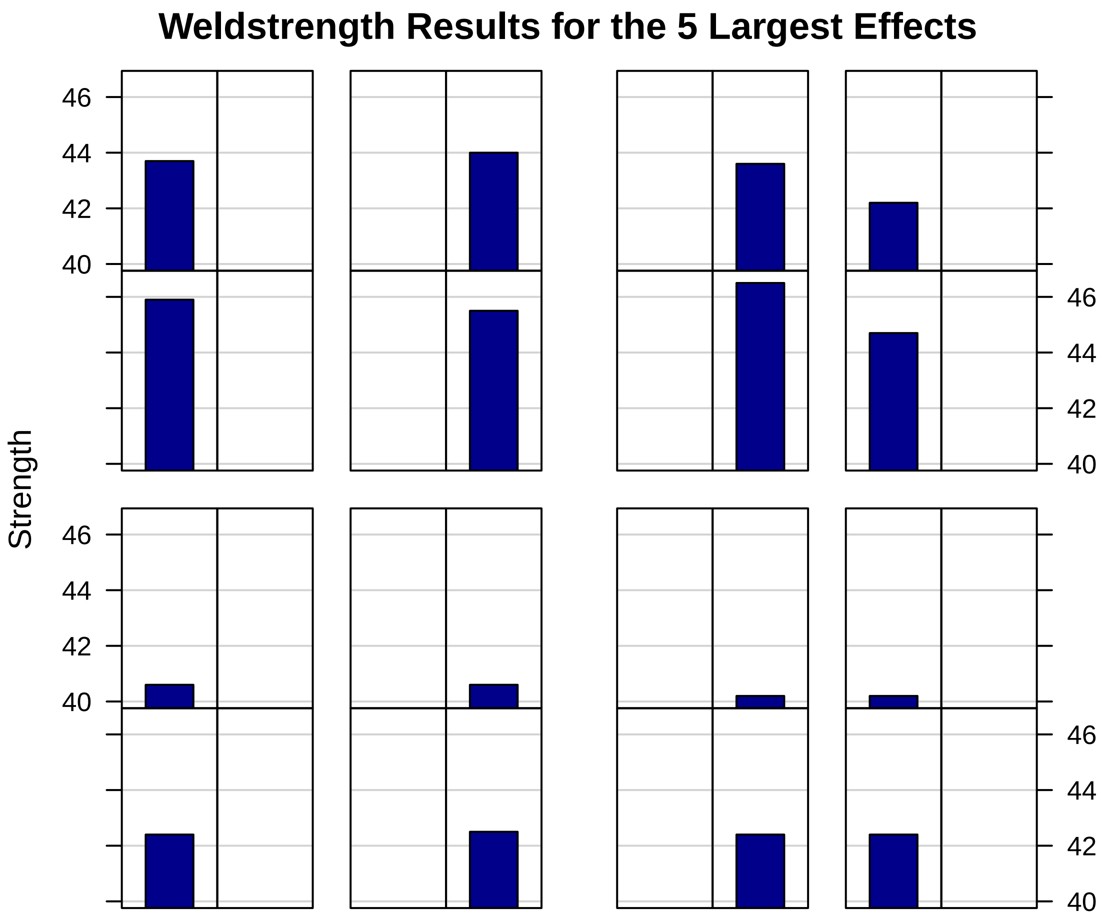
```
## (Intercept)        Rods      Drying    Material   Thickness       Angle
##       43.54        0.40        2.15       -3.10        0.13        0.05
##     Opening     Current      Method  Preheating
##       -0.40        0.15       -0.15       -0.37
```

We see that the Drying and Material factors have by far the largest magnitude (standardized) coefficients, that Rods, Opening, and Preheating have marginal and about equal in magnitude effects, and that the remaining 4 factors seem likely to be nothing more than experimental noise. It seems reasonable to view a projection onto the 5 largest factors. Because the Drying and Material effects should be easy to see, I display these vertically and the remaining 3 smaller effects horizontally, so that the results can be more easily compared across the rows.

```
strucplot(~Strength|Rods*Drying*Material*Opening*Preheating, xyLay = c(1,4,5),
data = weldstrength, panel= panel.bars,
main = "Weldstrength Results for the 5 Largest Effects")
```

```
##
## PLOT STRUCTURE
##
```

```
## _Horizontal_
## Rods:  0  1
## Opening:  0  1
## Preheating:  0  1
##
## _Vertical_
## Drying:  0  1
## Material:  0  1
```



The positive Drying effect is evident in the consistent increase in strength from the first to second row and third to fourth. Similary for the negative Material effect in the reduction from the top 2 rows (Material = 0) to the bottom 2 (Material =1).

The remaining 3 factors seem to have negligible effect except for the top right corner of the design. This is where the Material is "0" and Preheating is "1". At this condition, there are (at least) 2 equivalent ways of describing what is seen in the display: either weld strength decreases by about 2 units (about 5%) when Opening increases and/or when Rods decreases. The inability to say which – or even to say whether it could just be experimental noise – is due to the design's "parsimony" and the resulting inherent "confounding" of various effects. See (Box, Hunter, and Hunter 2005) for a fuller discussion of such matters.

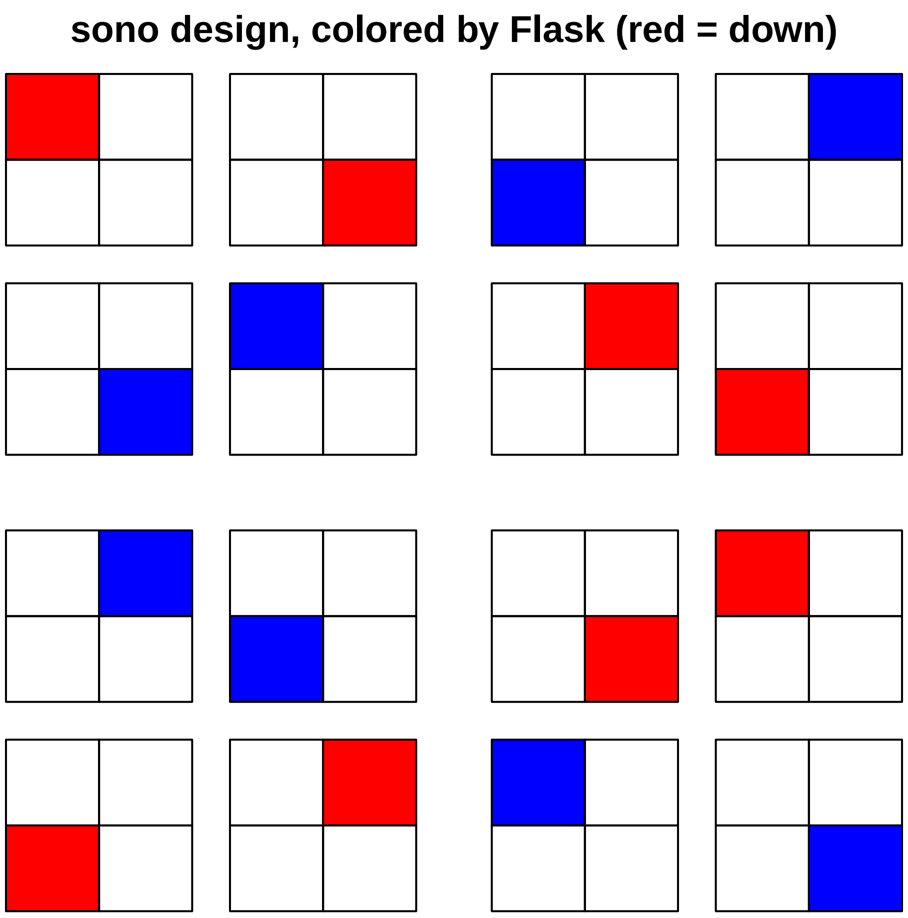
However, as that discussion emphasizes, screening designs such as this are not meant nor expected to be conclusive. Rather, they enlist the aid of Mother Nature in whittling down a large list of possible sources of variability in a phenomenon under study to those few with greatest impact. These then would need to be further studied in more focused designs. The interpretation here seems reasonable and consistent with such a program. What is relevant for our purposes is that it was simply derived without recourse to complex statistical arguments from a minimal statistical analysis and effective data visualization. In many applications of experimental design especially in industry, such simplicity is a virtue.

Coding by Groups

One can also preserve real estate and/or more clearly show patterns by using the `groups` parameter to code a factor via color or line type instead of as a conditioning factor of the display. I use the sono data set in the faraway package to illustrate this. This is a 7 factor, 2 level fractional factorial. First, I show the design using a grouping variable to color the panels by one of the factors. Note that to do this, the groups must be explicitly given in the call since there is no data argument in which they can be looked up for the data.frame method.

```r
strucplot(sono[,-c(1,8)],
        groups=sono$Flask,
        col=c("red","blue"),
        main = "sono design, colored by Flask (red = down)"
  )
```

```
##
## PLOT STRUCTURE
##
## _Horizontal_
## Molarity:  down  up
## Solute:  down  up
## pH:  down  up
##
## _Vertical_
## Gas:  down  up
## Water:  down  up
## Horn:  down  up
```

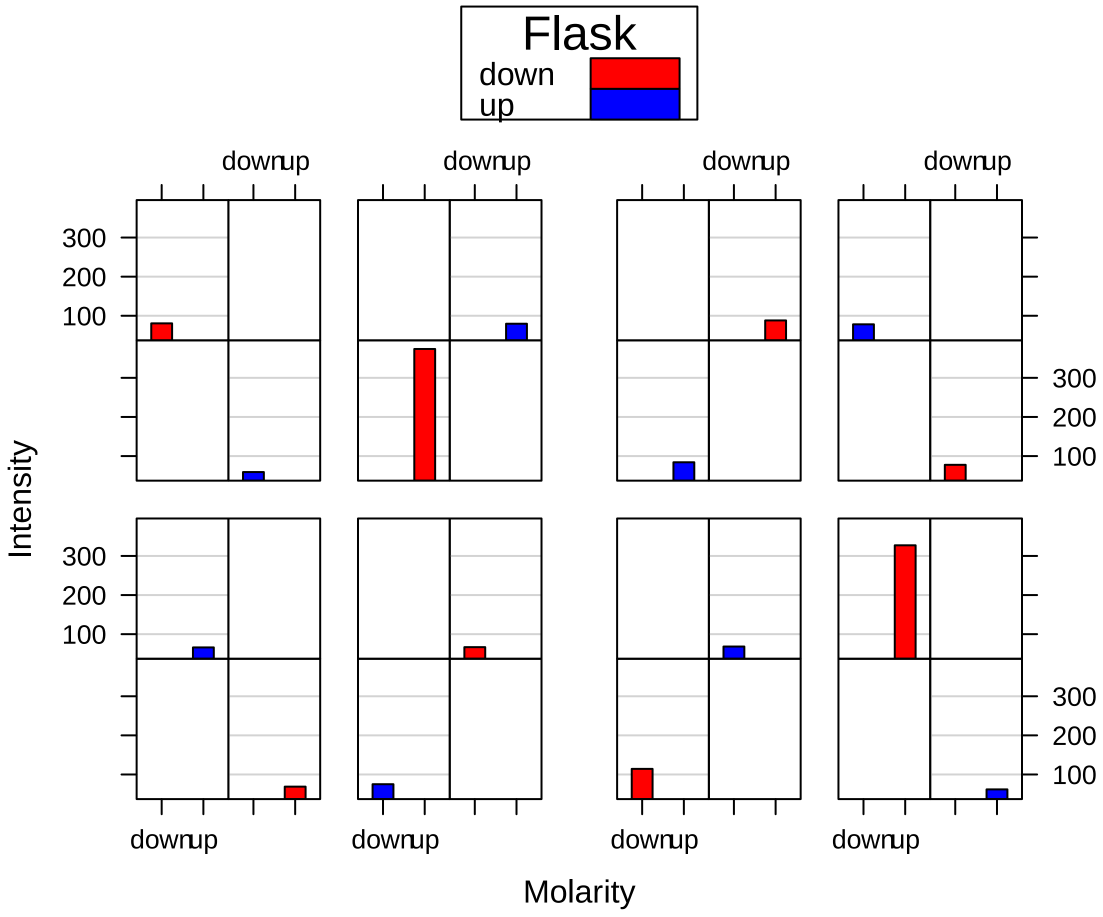


Using color nicely portrays the balance in the design. It also reduces the number of panels by half, from 128 to 64. When plotting the response, putting one of the variables on the x axis effects a further reduction by half from 64 to 32. To illustrate this, the following code plots the response ("Intensity") versus Molarity in each panel, still coloring by Flask. A key is used to give the group coding. Note that one may have to fiddle a bit to do this, as lattice uses different sets of parameters to control key features from those encoding the graphical elements when the defaults are not used, as here.

```
strucplot(Intensity~Molarity|Solute*pH*Gas*Water*Horn, groups =Flask,
          data = sono,
          panel=panel.superpose,
          par.settings = list(superpose.polygon = list(col=c("red","blue"))),
          fill = c("red","blue"),
          auto.key = list(points=FALSE,rectangles=TRUE,
              title = "Flask",lines.title =1.5, border =TRUE,cex.title = 1.5),
          panel.groups = function(x,y,fill,...) panel.bars(x=x,y=y,col=fill))
```

```
##
## PLOT STRUCTURE
##
## _Horizontal_
## Solute:  down  up
## pH:  down  up
## Gas:  down  up
##
```

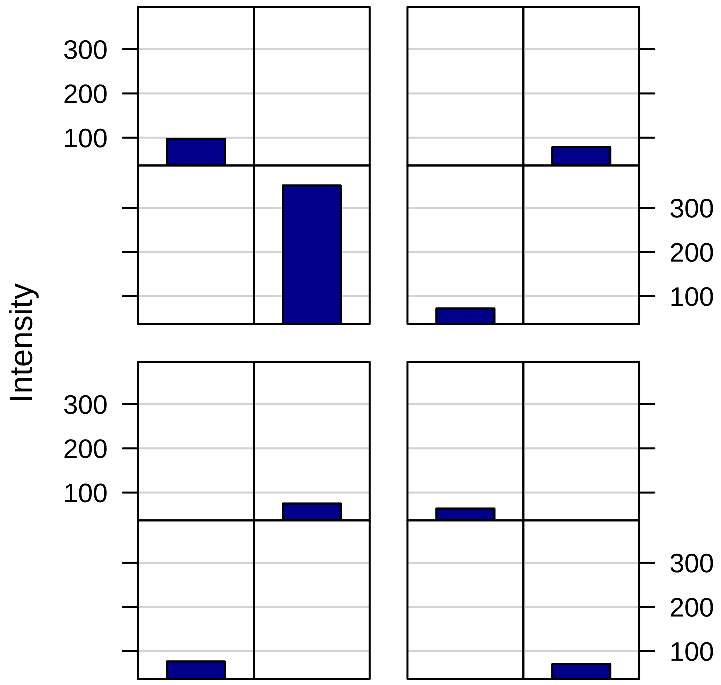
```
## _Vertical_
## Water:  down  up
## Horn:  down  up
```



This plot clearly shows that except for the two combinations when Molarity was “up”,Solute was “down”, pH was “up”, and Flask was “down”, Intensity was about 80; for these two combinations, Intensity was about 350. Unfortunately, due to confounding in the design, it is not possible to unequivocally untangle the effects of these four factors. This can be seen by showing the full 16 panel plot in just these four factors.

```
strucplot(~Intensity|Molarity*Solute*pH*Flask, data = sono,
          panel=panel.bars
)
```

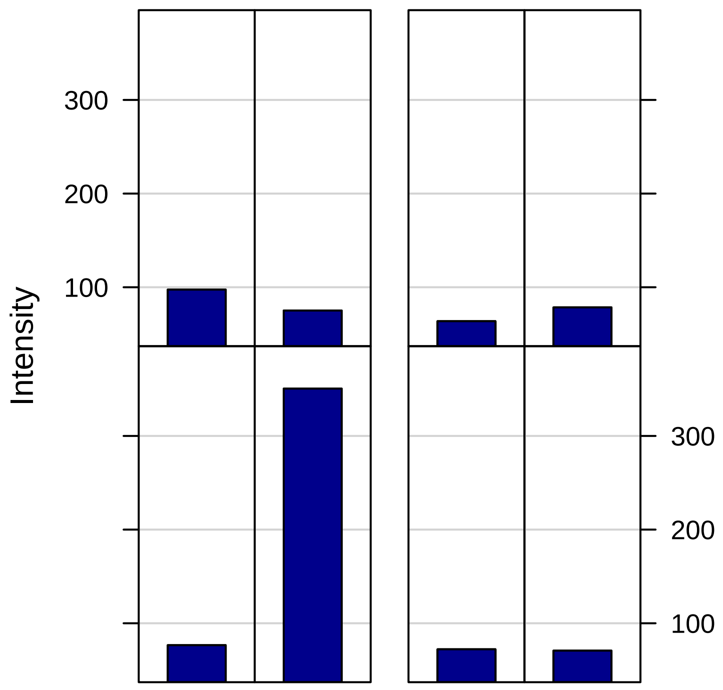
```
##
## PLOT STRUCTURE
##
## _Horizontal_
## Molarity:  down  up
## Solute:  down  up
##
## _Vertical_
## pH:  down  up
## Flask:  down  up
```



There are only 8 unique combinations (it's a half-fraction in these 4 factors) and only one combination shows an effect. This means that any three of the factors suffice to characterize the effect (i.e., main effects are confounded with 3 factor interactions). So for example, dropping Flask:

```
strucplot(~Intensity|Molarity*Solute*pH, data = sono, panel=panel.bars)
```

```
##
## PLOT STRUCTURE
##
## _Horizontal_
## Molarity:  down  up
## Solute:  down  up
##
## _Vertical_
## pH:  down  up
```



So maybe Flask has no effect, and it's just this combination of Molarity, Solute, and pH that causes the high intensity. Only subject matter expertise and/or further experimentation can resolve this.

Incidentally, it is perhaps worth again noting that while all of this can be elucidated in a formal statisical analysis, say by multiple regression, the visualization is both simple and sufficient to reveal what the data have to say. Of course, once one has gotten matters down to three or four factors, strip labels no longer take up so much space, so there is not as compelling a reason to remove them. Even so, some may prefer these "cleaner" displays.

When there are relatively few panels containing complex displays, removing the strips may still reclaim visual real estate that faciltates comparisons and better engages our pattern recognition machinery. To illustrate, I show how to get `strucplot` to make a somewhat more complex display that requires a bit of "fiddling": a histogram of the data with an overlaid kernel density plot.[1] The fiddling is required because `strucplot` is a wrapper for `xyplot`, which requires a y response versus an x covariate. Axis scales, labels, etc. for the panels are all constructed from the x, y data based on this paradigm. However histograms and densities are constructed from a single x variable by panel functions, so there is no y variable. So to construct a display like a histogram in which the panel function constructs the "y" values, one must use a formula of the form `x~x` and set y limits explicitly, which typically requires several attempts to get it right.[2] Here's the code illustrating this, using constructed data.

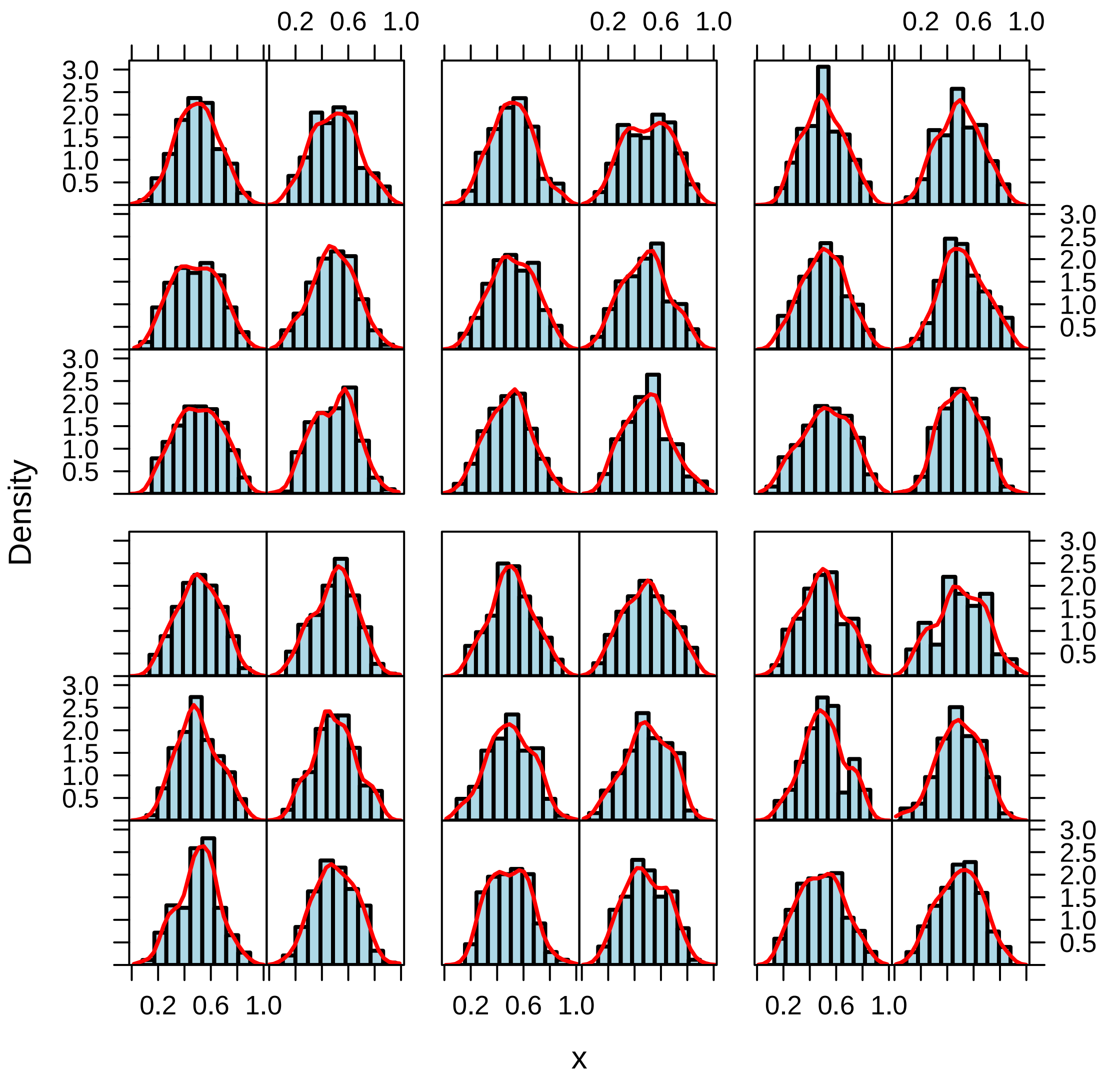
```
## Create a data frame with 4 factors, 2 at 3 levels each and 2 and 2 levels each,
## with 200 random beta(4,4) responses for each of the 36 settings

##Set a random seed for reproducibility and generate the data
set.seed(2222)
d <- data.frame(x=rbeta(7200,4,4),
    expand.grid(a= 0:1, b= 1:3, c=letters[1:3], d = c("A","B"))[rep(1:36,e=200),])
```

[1]Note that lattice's `densityplot()` and `histogram()` functions – and their corresponding panel functions – are already perfectly capable of producing such displays. But these are high level interfaces that bypass `xyplot`.

[2]The `~x` formula works only when no x axis is needed for the display, e.g. for strip plots or box plots of a vector of responses, or barplots of mean responses.

```
## Construct the display
strucplot(x~x|.,data = d, ylim = c(0,3.2), ylab = "Density",
      col= "lightblue", col.line = "red",lwd=2,
      panel = function(x,y,col, col.line, ...){
      panel.histogram(x, col=col,breaks=NULL, ...)
      panel.densityplot(x, plot.points= FALSE,col= col.line,... )
       }
)
```

```
##
## PLOT STRUCTURE
##
## _Horizontal_
## a:   0   1
## b:   1   2   3
##
## _Vertical_
## c:   a   b   c
## d:   A   B
```



The absence of strip labels allows the eye to easily compare the complex patterns among the panels (although there's nothing much to see here, of course). Strip labels within the panels would make this much more difficult, aside from reducing the amount of visual real estate.

Summary

Using regular layouts and simple legends instead of layers of strip labels to encode factor settings in trellis graphics often helps to better reveal and apprehend informative patterns in the data. The **`strucplot`** methods described here are a simple extension of the lattice package's `xyplot` framework that allows this to be easily done. The examples show how this may be useful, especially for the factorial designs widely used in industrial applications. But multipanel conditioning – aka small multiples – is a broadly applicable (and underutilized) data display paradigm; `strucplot` is meant to make stripless versions available for most situations in which such displays are contemplated. I hope that this will not only be useful in its own right, but that it also encourages greater application of trellis-type multivariate visualization for the many data analysis tasks where they can be of value.

References

Box, George E.P., J. Stuart Hunter, and William G. Hunter. 2005. *Statistics for Experimenters, Second Edition.* Hoboken, NJ: John Wiley & Sons.

Cleveland, Willam S. 1993. *Visualizing Data.* Summit, NJ: Hobart Press.

Murrell, Paul. 2011. *R Graphics, Second Edition.* The R Series, ISBN 978-1439831762. Boca Raton: CRC Press/Chapman & Hall.

R Core Team. 2015. *R: A Language and Environment for Statistical Computing.* Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Sarkar, Deepayan. 2008. *Lattice: Multivariate Data Visualization with R.* New York: Springer. http://lmdvr.r-forge.r-project.org.

Tufte, Edward R. 1983. *The Visual Display of Quantitative Information.* Cheshire, CT: Graphics Press.