The Strucplot Framework—Visualizing Multi-way Contingency Tables

by David Meyer
Wirtschaftsuniversität Wien, Austria
David.Meyer@R-project.org

September 19, 2005

1 Framework Overview

The strucplot framework in the R package **vcd** visualizes multi-way contingency tables and integrates techniques such as mosaic displays, association plots, and sieve plots. The main idea is to visualize the tables' cells arranged in rectangular form. For multi-way tables, the variables are nested into row and columns using recursive conditional splits, given the margins. The result is a 'flat' representation that can be visualized in similar ways than a two-dimensional table. This principle defines a class of conditional displays with still many parameters left such as:

- the content of the tiles
- the split direction for each dimension
- the graphical parameters of the tiles' content
- the spacing between the tiles
- the labeling of the tiles

The document at hand gives an introduction to the framework, whereas labeling and shading issues are described in separate vignettes.

The strucplot framework is highly modularized; Figure 1 shows the hierarchical relationship between the various components. On the lowest level, there are several groups of workhorse and parameter functions that directly or indirectly influence the final appearance of the plot. The workhorse functions (*`struc_foo()`*, *`labeling_foo()`*, and *`legend_foo()`*) directly produce graphical output, the parameter functions (*`spacing_foo()`* and *`shading_foo()`*) compute graphical parameters used by the others. The *`struc_foo()`* functions implement the core functionality, creating the tiles and their content. On the second level, a suitable combination of these functions is passed as "hyperparameters" to `strucplot()`. These central function sets up the graphical layout using grid viewports (see Figure 2), and coordinates the specified core, labeling, shading, and spacing functions to produce the plot. On the third level, we provide several high-level functions such as `mosaic()`, `sieve()`, `assoc()`, and `doubledecker()` which conveniently interface `strucplot()` through sensible parameter defaults and support for model formulas. Finally, on the fourth level, there are 'related' **vcd** functions (such as `pairs.table()` and `cotabplot()`) combining single plots of the strucplot framework into more complex displays.

2 Mosaic, Association, and Sieve Plots

As an example, consider the '`HairEyeColor`' data containing two polytomous variables (hair and eye color), as well as one (artificial) dichotomous variable (sex, i.e. gender). The 'flattened'

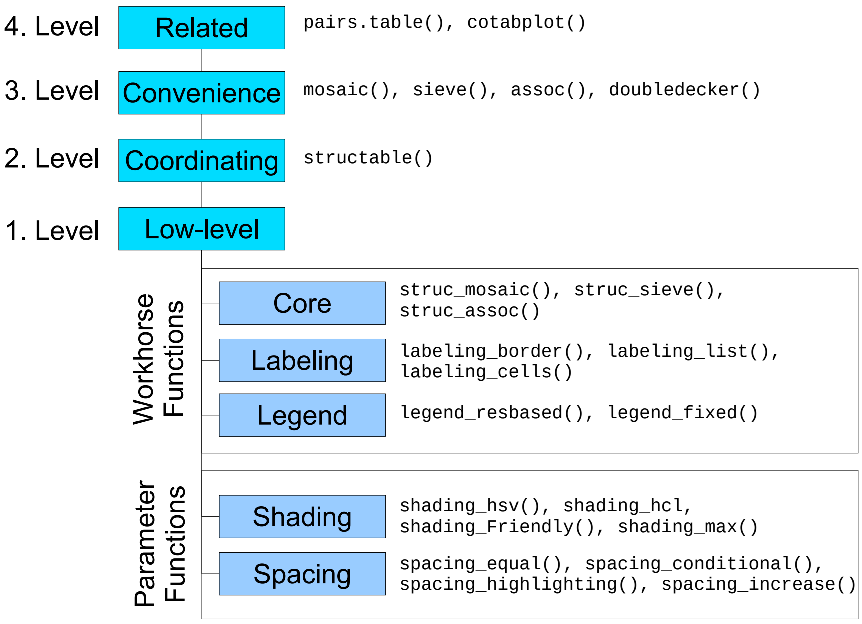

Figure 1: Components of the strucplot framework.

contingency table can be obtained using the `structable()` function (quite similar to `ftable()` in base R, but allowing the specification of split directions):

```
> (hec <- structable(aperm(HairEyeColor)))

             Eye Brown Blue Hazel Green
Sex    Hair
Male   Black        32   11    10     3
       Brown        38   50    25    15
       Red          10   10     7     7
       Blond         3   30     5     8
Female Black        36    9     5     2
       Brown        81   34    29    14
       Red          16    7     7     7
       Blond         4   64     5     8
```

Let us first visualize the contingency table by the means of a mosaic plot (Hartigan and Kleiner, 1984) which is basically an area-proportional visualization of (typically observed) frequencies, composed of tiles (corresponding to the cells) created by recursive vertical and horizontal splits of a square. Thus, the area of each tile is proportional to the corresponding cell entry *given* the dimensions of previous splits. Figure 3 depicts the effect of

```
> mosaic(hec)
```

equivalent to

```
> mosaic(~Sex + Eye + Hair, data = HairEyeColor)
```

The small bullets indicate zero entries in the corresponding cell. It is also possible to visualize the expected values instead of the observed values (see Figure 4):

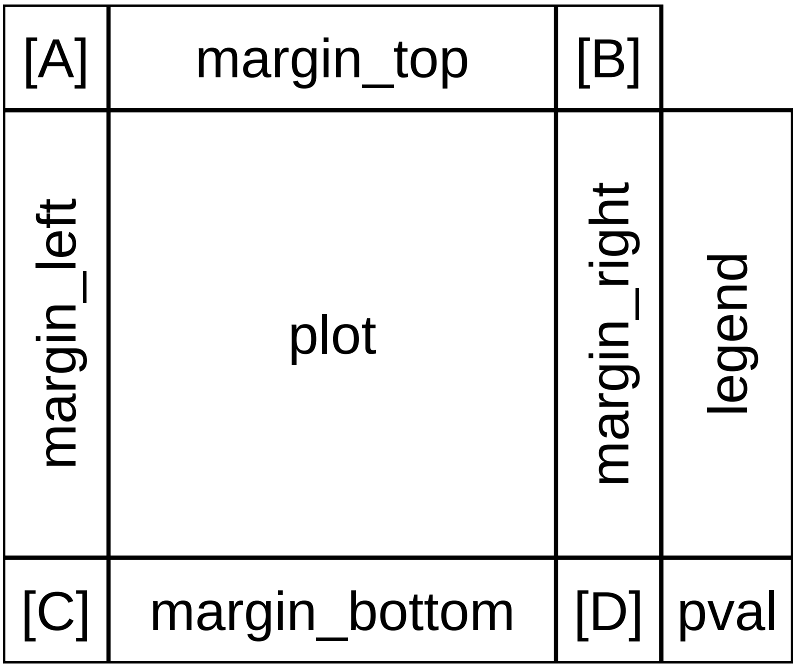

Figure 2: Viewport layout for strucplot displays with their names. [A] = "corner_top_left", [B] = "corner_top_right", [C] = "corner_bottom_left", [D] = "corner_bottom_right".

```
> mosaic(hec, type = "expected")
```

In order to compare observed and expected values, a sieve plot (Riedwyl and Schüpbach, 1994) could be used (see Figure 5):

```
> sieve(hec)
```

Alternatively, we can directly inspect the residuals. The Pearson residuals (standardized deviations of observed from expected values) are preferably visualized using association plots (Cohen, 1980). In contrast to `assocplot()` in base R, **vcd**'s `assoc()` function scales to more than two variables (see Figure 6):

```
> assoc(hec, compress = FALSE)
```

The `compress` argument keeps distances between tiles equal for better comparison.

For both mosaic plots and association plots, the splitting of the tiles can be controlled using the `split_vertical` argument (default: alternating splits starting with a vertical one).

```
> mosaic(hec, split_vertical = c(TRUE, TRUE, FALSE))
```

For compatibility with `mosaicplot()` in base R, the `mosaic()` function also allows the use of a `"direction"` argument taking a vector of `"h"` and `"v"` characters (see Figure 7):

```
> mosaic(hec, direction = c("v", "h", "v"))
```

By a suitable combination of splitting, spacing, and labeling settings, the funtions provided by the strucplot framework can be customized in a quite flexible way. For example, it was almost trivial to implement doubledecker plots—`doubledecker()` is really just a wrapper for `mosaic()`, setting the right defaults. Figure 8 shows a doubledecker plot of the '`Titanic`' data, explaining the probability of survival ('Survived') by Age, given Sex, given Class. It is created by:

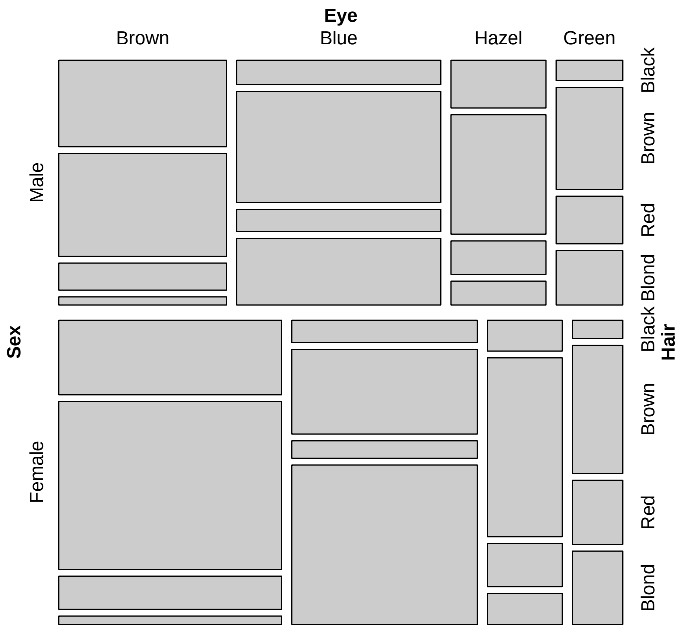

Figure 3: Mosaic plot for the 'HairEyeColor' data.

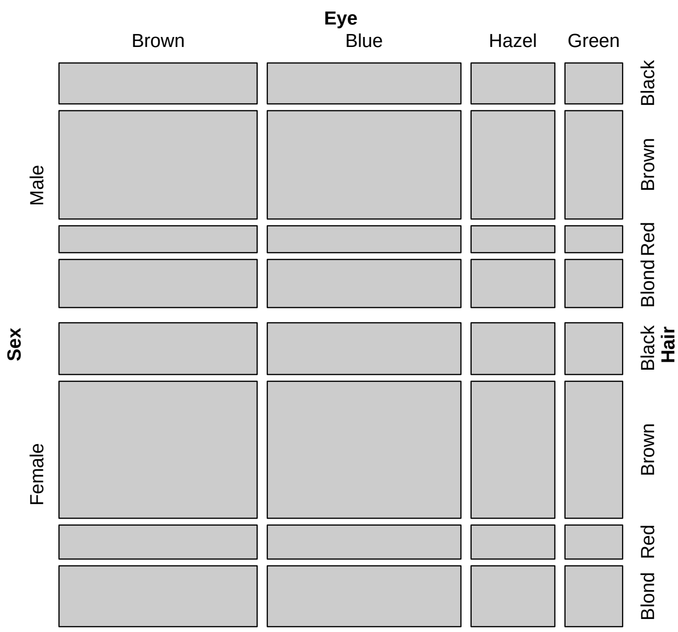

Figure 4: Mosaic plot for the 'HairEyeColor' data (expected values).

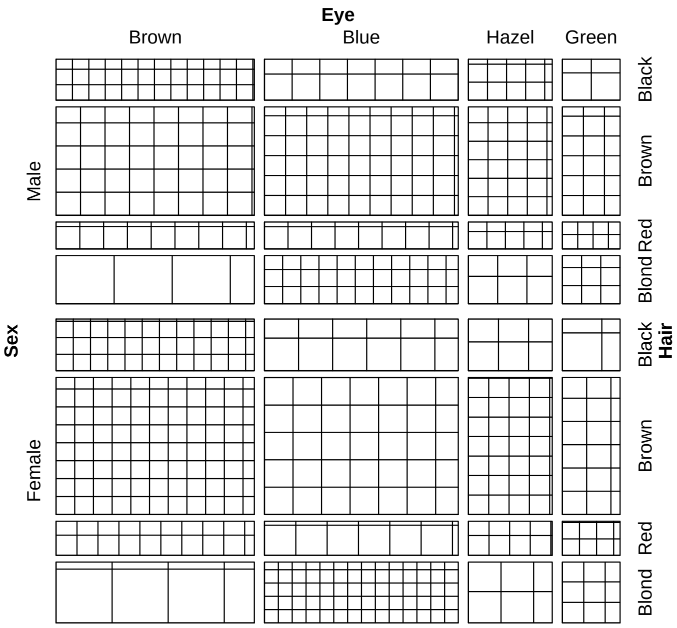

Figure 5: Sieve plot for the 'HairEyeColor' data visualizing simultaneously observed and expected values.

```
> doubledecker(Titanic)
```

equivalent to:

```
> doubledecker(Survived ~ Class + Sex + Age, data = Titanic)
```

3 Conditional and partial views

So far, we have visualized full tables. For objects of class `table`, conditioning on levels (i.e., choosing a table subset for fixed levels of the conditioning variable(s)) is simply done by indexing. However, subsetting `"structable"` objects is more restrictive because of their inherent conditional structure. Since the variables on both the row and the columns side are nested, conditioning is only possible "outside-in":

```
> hec

             Eye Brown Blue Hazel Green
Sex    Hair
Male   Black        32   11    10     3
       Brown        38   50    25    15
       Red          10   10     7     7
       Blond         3   30     5     8
Female Black        36    9     5     2
       Brown        81   34    29    14
       Red          16    7     7     7
       Blond         4   64     5     8
```

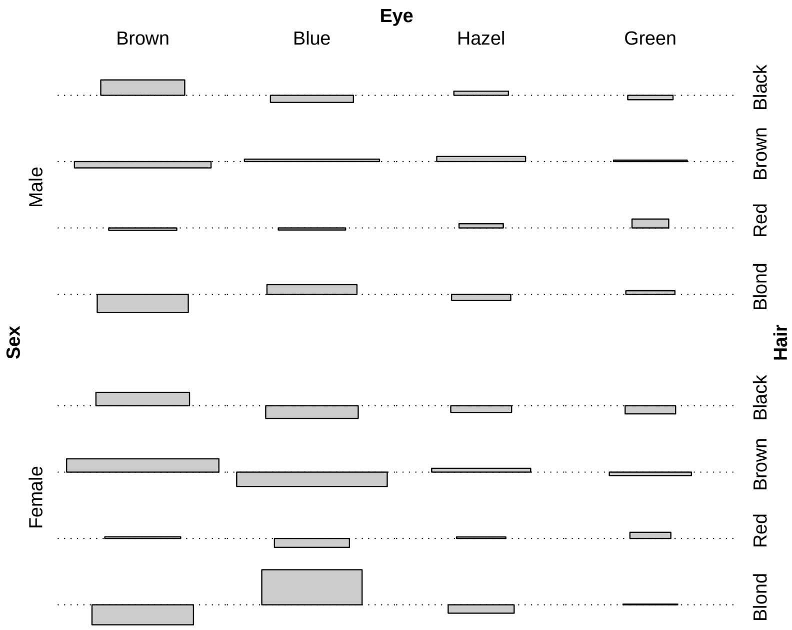

Figure 6: Association plot for the '`HairEyeColor`' data.

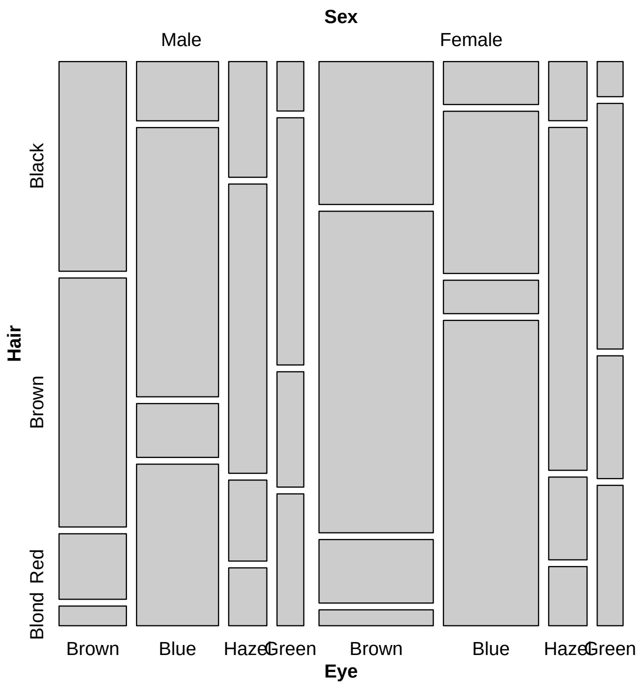

Figure 7: Mosaic plot for the '`HairEyeColor`' data—alternative splitting.

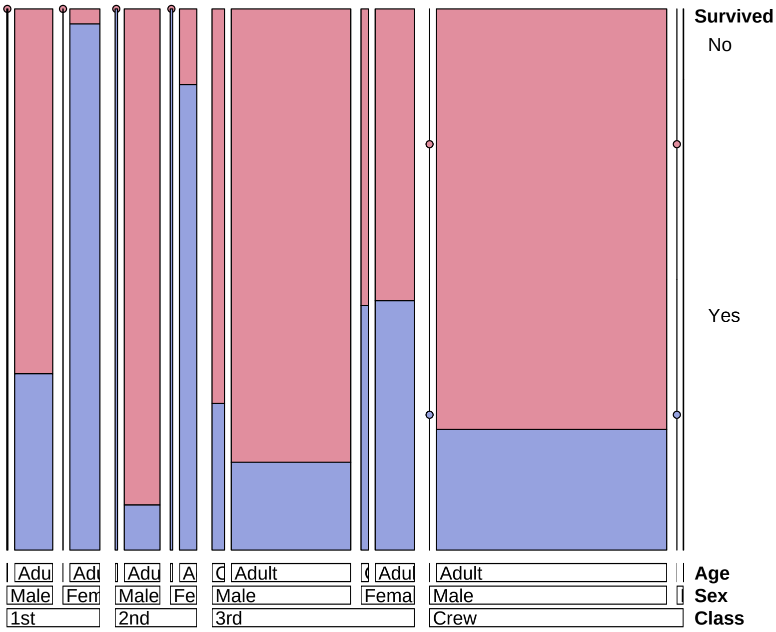

Figure 8: Doubledecker plot for the 'Titanic' data.

```
> hec[["Male", ]]

      Eye Brown Blue Hazel Green
Hair
Black        32   11    10     3
Brown        38   50    25    15
Red          10   10     7     7
Blond         3   30     5     8

> hec[[c("Male", "Brown"), ]]

Eye Brown Blue Hazel Green

       38    50    25    15

> hec[["Male", "Green"]]

Hair
Black   3
Brown  15
Red     7
Blond   8
```

Now, there are several ways for visualizing conditional independence structures. The "brute force" method is to draw separate plots for the strata. The following example compares the association between hair and eye color, given gender, by using subsetting on the flat table and the **grid** package's viewport framework to visualize the two groups besides each other:

```
> pushViewport(viewport(layout = grid.layout(ncol = 2)))
> pushViewport(viewport(layout.pos.col = 1))
```

```
> mosaic(hec[["Male"]], margins = c(top = 2, 0), sub = "Male", newpage = FALSE)
> popViewport()
> pushViewport(viewport(layout.pos.col = 2))
> mosaic(hec[["Female"]], margins = c(top = 2, 0), sub = "Female", newpage = FALSE)
> popViewport(2)
```

Note the use of the `margins` argument: it takes a vector with up to four values whose unnamed components are recycled, but "overruled" by the named arguments. Thus, in the example, only the top margin is set to 2 lines, and all other to 0. This idea applies to almost all vectorized arguments in the strucplot framework (with `split_vertical` as a prominent exception).

Since mosaic displays are "conditional plots" by definition, we can also use one single mosaic for stratified plots. The formula interface of `mosaic()` allows the specification of conditioning variables (see Figure 10):

```
> mosaic(~Hair + Eye | Sex, data = hec, split_vertical = TRUE)
```

The effect of using this kind of formula is that conditioning variables are permuted before the conditioned variables in the table, and that `spacing_conditional()` is used as default to better distinguish conditioning from conditioned dimensions. This spacing uses equal space between tiles of conditioned variables, and increasing space between tiles of conditioning variables. Note, however, that the plots in the "pseudo-strata" are distorted since they are not corrected for the marginal distribution(s) of the conditioning variables.

The `cotabplot()` function does a much better job on this task: it arranges stratified strucplot displays in a lattice-like layout, conditioning on variable *levels*. The plot in Figure 11 shows Hair and Eye color, given Sex:

```
> cotabplot(~Hair + Eye | Sex, data = hec, labeling_args = list(abbreviate = c(Eye = 2)))
```

The `labeling_args` argument ensures that 'Eye' levels are abbreviated (See the vignette on labeling for detailed information).

Another high-level function for visualizing conditional independence models is `pairs.table()` (`pairs.structable()`), that is, the S3-method for objects of class `"table"` (`"structable"`) of the `pairs()` generic. In contrast to `cotabplot()` which is conditioning on variable *levels*, `pairs.table()` is conditioning on the variables themselves, creating partial views of the table. The function produces a matrix having strucplot displays in the off-diagonal cells, and the variable names (with, optionally, univariate statistics) in the diagonal cells. Figure 12 shows a pairs display with mosaic plots visualizing mutual independence in the lower triangle, association plots for the same in the upper triangle, and bar charts in the diagonal.

```
> pairs(hec, upper_panel = pairs_assoc, space = 0.3, diag_panel_args = list(rot = -45,
+     just_leveltext = c("left", "bottom")))
```

In plots produced by `pairs.table()`, each cell's row and column define two variables X and Y used for the specification of four different types of independence: '`pairwise`', '`total`', '`conditional`' and '`joint`'. The pairwise mosaic matrix shows bivariate marginal relations between X and Y, collapsed over all other variables. The total independence mosaic matrix shows mosaic plots for mutual independence, i.e., for marginal and conditional independence among all pairs of variables. The conditional independence mosaic matrix shows mosaic plots for marginal independence of X and Y, given all other variables. The joint independence mosaic matrix shows mosaic plots for joint independence of all pairs (X, Y) of variables from the others.

Since the matrix is symmetric, the upper and lower parts can independently be used to display different types of independence models, or different strucplots displays (mosaic, association, or sieve plots). The available core functions (`pairs_assoc()`, `pairs_mosaic()`, and `pairs_sieve()`) are just simple wrappers to `assoc()`, `mosaic()`, and `sieve()`, respectively. Obviously, seeing patterns in strucplot matrices becomes increasingly difficult with higher dimensionality. Typically, therefore, this plot is used with a suitable residual-based shading (described in a separate vignette).

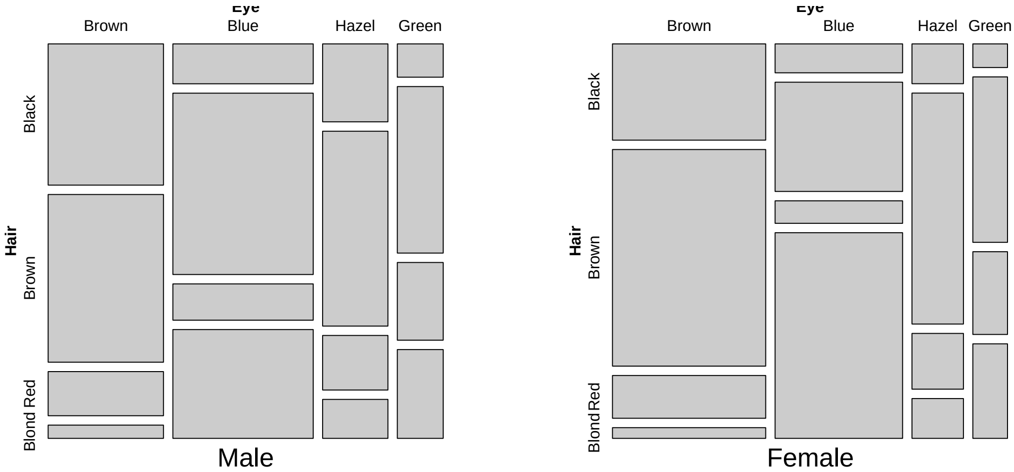

Figure 9: Distribution of hair and eye color, given gender.

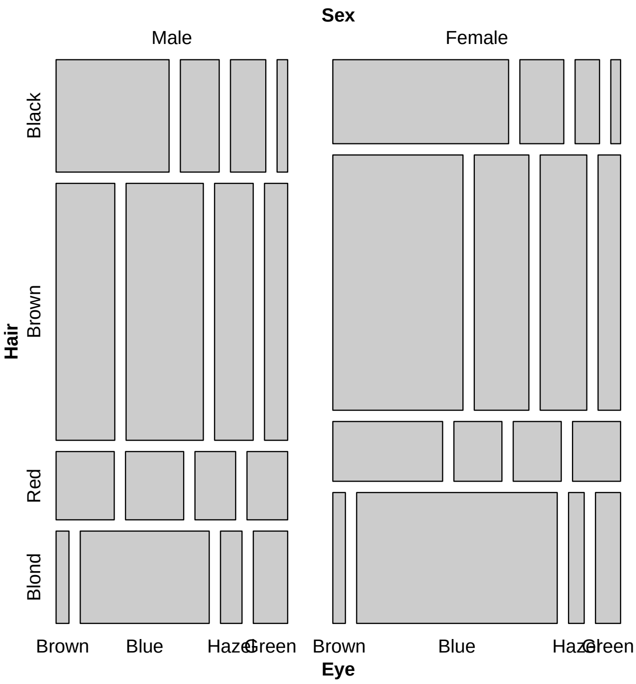

Figure 10: Mosaic plot for conditional independence structures.

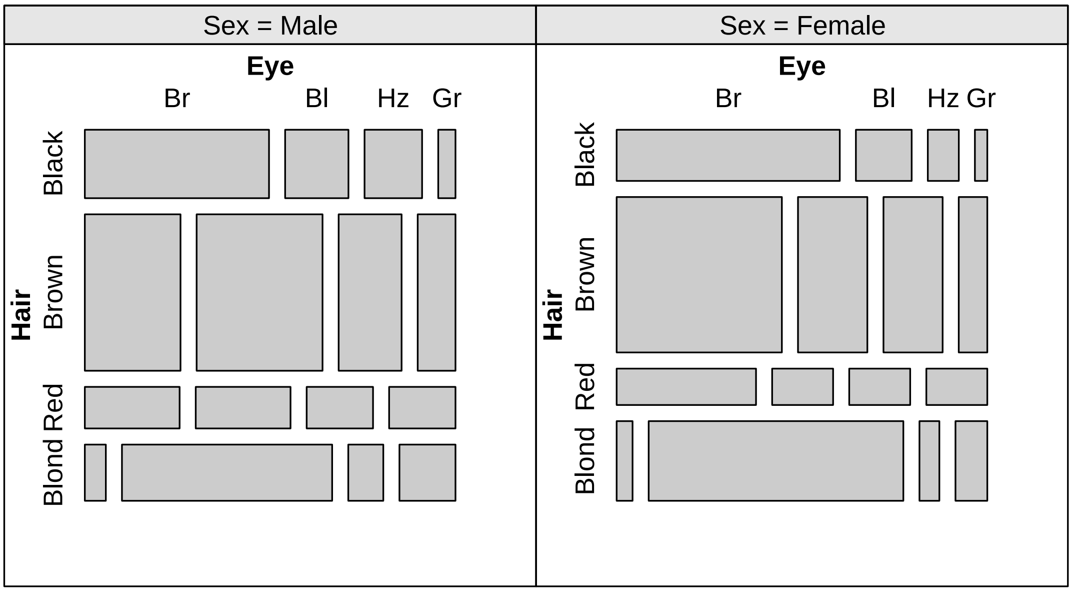

Figure 11: Conditional tabular plot for the 'HairEyeColor' data.

4 Interactive plot modifications

All strucplot core functions are supposed to produce conditional hierarchical plots by the means of nested viewports, corresponding to the provided splitting information. Thus, at the end of the plotting, each tile is associated with a particular viewport. Each of those viewports has to be conventionally named, enabling other strucplot modules, in particular the labeling functions, to access specific tiles after they have been plotted. The naming convention for the viewports is:

`cell:`*`[Variable 1]=[Level 1],[Variable 2]=[Level 2] ...`*

Clearly, these names depend on the splitting. The following example shows how to access parts of the plot after it has been drawn (see Figure 13):

```
> mosaic(~Hair + Eye, data = hec, pop = FALSE)
> seekViewport("cell:Hair=Blond")
> grid.rect(gp = gpar(col = "red", lwd = 4))
> seekViewport("cell:Hair=Blond,Eye=Blue")
> grid.circle(r = 0.2, gp = gpar(fill = "cyan"))
```

Note that the viewport tree is removed by default. Therefore, the `pop` argument has to be set to `FALSE` when viewports shall be accessed.

In addition to the viewports, the main graphical elements get names following a similar construction method. This allows to change graphical parameters of plot elements *after* the plotting (see Figure 14):

```
> assoc(Eye ~ Hair, data = hec, pop = FALSE)
> getNames()[1:6]
```

```
[1] "GRID.GROB.4724"            "rect:Hair=Black,Eye=Brown" "GRID.GROB.4725"
[4] "rect:Hair=Brown,Eye=Brown" "GRID.GROB.4726"            "rect:Hair=Red,Eye=Brown"
```

```
> grid.edit("rect:Hair=Blond,Eye=Blue", gp = gpar(fill = "red"))
```

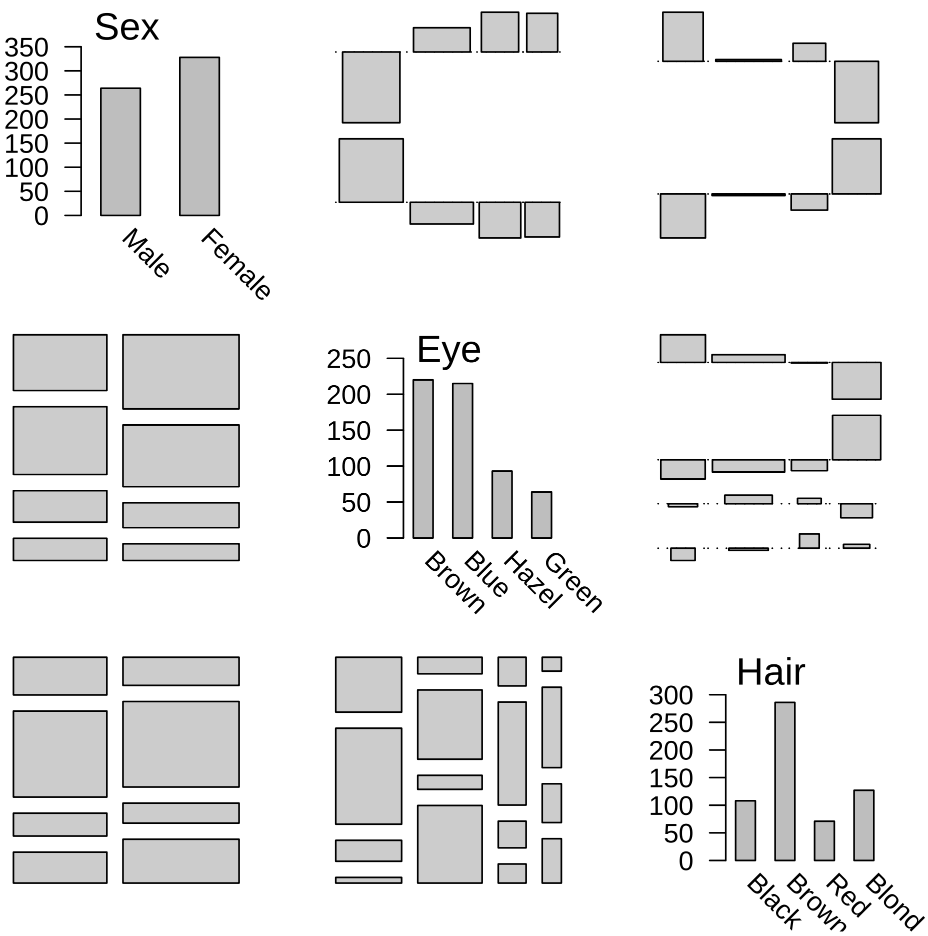

Figure 12: Pairs plot for the ‘`HairEyeColor`’ data.

References

Cohen, A. (1980). On the graphical display of the significant components in a two-way contingency table. *Communications in Statistics—Theory and Methods*, A9:1025–1041.

Hartigan, J. and Kleiner, B. (1984). A mosaic of television ratings. *The American Statistician*, 38:32–35.

Riedwyl, H. and Schüpbach, M. (1994). Parquet diagram to plot contingency tables. In Faulbaum, F., editor, *Softstat ’93: Advances in Statistical Software*, pages 293–299, New York. Gustav Fischer.

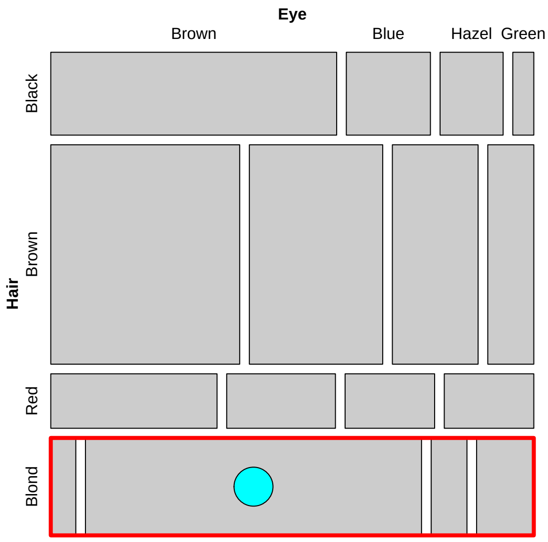

Figure 13: Adding elements to a mosaic plot after drawing.

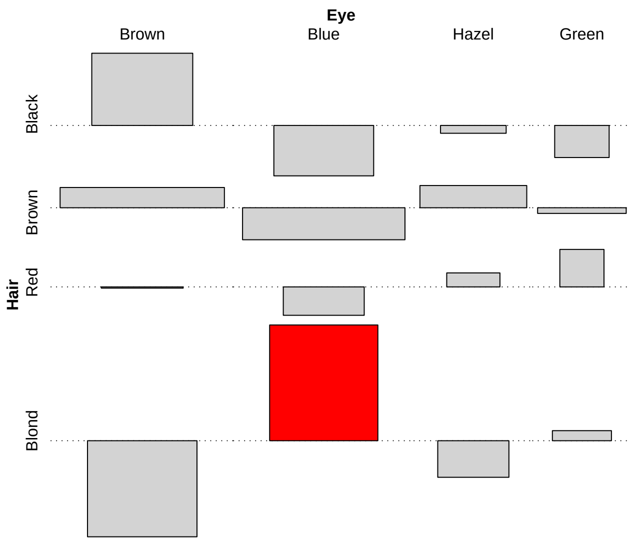

Figure 14: Changing graphical parameters of elements after drawing.