R Package `CellPlot` User Manual

Sven E. Templer, Robert Sehlke
`rsehlke@age.mpg.de`

September 17, 2015

Contents

1 Introduction

1.1 Description

CellPlot is a GNU R package with plot methods for the integrated visualisation of functional enrichment and expression data. It can be used to display commonly used gene ontology (GO) enrichment analysis from differential gene expression.

1.2 Installation & Development

You can install the package to use it in any `R` session. The stable versions will be published on the comprehensive R archive network (CRAN) servers. The most recent version can be found on github.com/dieterich-lab/CellPlot. With the `devtools` package, it is easy to install:

```
> install.packages("devtools")
> library(devtools)
> install_github(repo='dieterich-lab/CellPlot')
```

Once installed, you need to load the package into the current session to access the functions and help pages:

```
> library(CellPlot)
```

General help is available at the manual pages of the functions or package, e.g.:

```
> ?CellPlot
```

2 DEG and GO enrichment analysis

The following code represents a simple analysis workflow to analyse/test for differentially regulated genes from two conditions, perform GO enrichment analysis and merge the data to a `data.frame`. The example data therefore is taken from the `multtest` package, available from the platform. It contains microarray gene expression data from leukemia study of Golub et al. (1999), processed as described in Dudoit et al. (2002).

Differential gene expression was performed using a Student t-test to compare the two groups. GO annotation was done with the `annotate` and `hu6800.db` packages from Bioconductor. The `topGO` package was utilized to perform a GO enrichment test based on Fisher's exact test. Data of the DEG and GO enrichment was merged with the provided go.enrich function. The code to create the data object is shown in the examples section. For help about the data set also read the manual page of the resulting and implemented data set (`?golubstat`):

```
> ### Differential gene expression
> ### and GO enrichment analysis
> ### of the golub dataset:
>
> ## install dependencies
> source("http://bioconductor.org/biocLite.R")
> biocLite("hu6800.db")
> biocLite("multtest")
> biocLite("topGO")
> ## load dependencies
> library(CellPlot)
> library(plyr)
> library(multtest)
> library(annotate)
> library(hu6800.db)
> library(topGO)
> ## data import
> data(golub)
> ## differential gene expression testing
> golub.test <- function (e, i, n) {
+   i <- as.logical(i)
+   a <- split(e[,i],seq(nrow(e)))
+   b <- split(e[,!i],seq(nrow(e)))
+   fun.map <- function (a, b) {
+     t <- t.test(a, b)
+     t$fc <- t$estimate[1]/t$estimate[2]
+     suppressWarnings(t$fc <- log2(t$fc))
+     data.frame(
+       mean.aml = t$estimate[1],
+       mean.all = t$estimate[2],
+       log2fc  = t$fc,
+       p = t$p.value,
+       row.names = NULL)
+   }
+   s <- Map(fun.map, a, b)
+   s <- do.call(rbind, s)
+   s$p.adj <- p.adjust(s$p)
+   s <- data.frame(n, s, stringsAsFactors = FALSE)
+   s <- subset(s, !is.na(log2fc))
+   return(s)
+ }
> degdata <- golub.test(golub, golub.cl, data.frame(
+   gene = golub.gnames[,3], index = as.integer(golub.gnames[,1]),
+   stringsAsFactors = FALSE))
> ## gene ontology annotation
> goterms <- select(
+   hu6800.db, degdata$gene,
+   c("PROBEID","ALIAS","GO","ENSEMBL","ENTREZID"))
> goterms <- dlply(
+   subset(goterms, ONTOLOGY == "BP"), "PROBEID",
+   function (x) unique(x$GO))
> ## topGO object
> godata <- new(
+   "topGOdata", ontology = "BP", description = 'golub',
+   allGenes = setNames(degdata$p.adj, degdata$gene),
+   geneSelectionFun = function (allScore) { allScore <= 0.05 },
+   annotationFun = annFUN.gene2GO, gene2GO = goterms)
> ## gene ontology enrichment
> #golubstat <- go.enrich(godata, degdata, p.max = 1)
> golubstat <- go.enrich(godata, degdata)
```

A table like `degdata` from this example might also be obtained from DEG analysis with common tools as used in the Tuxedo pipeline (tophat/cuffdiff) and imported with `read.delim`. The resulting data object is available with our package. You can load it with the command:

```
> data("golubstat")
```

The columns are described in the help page, and the data structure can be investigated with the `str` and `head` functions, e.g.:

```
> ?golubstat
> str(head(subset(golubstat, !duplicated(genes))))
```

```
'data.frame':        6 obs. of  11 variables:
 $ id         : chr  "GO:0007068" "GO:0000710" "GO:0001675" "GO:0002358" ...
 $ term       : chr  "negative regulation of transcription dur..." "meiotic mismatch repair" "acroso
 $ annotated  : int  2 1 1 1 1 1
 $ significant: int  2 1 1 1 1 1
 $ expected   : num  0.05 0.03 0.03 0.03 0.03 0.03
 $ p          : num  0.00065 0.02571 0.02571 0.02571 0.02571 ...
 $ padj       : num  1 1 1 1 1 1
 $ loge       : num  3.69 3.51 3.51 3.51 3.51 ...
 $ genes      :List of 6
  ..$ : chr  "D26156_s_at" "L41870_at"
  ..$ : chr "U73737_at"
  ..$ : chr "U72342_at"
  ..$ : chr "M94633_at"
  ..$ : chr "U50928_at"
  ..$ : chr "M55150_at"
 $ deg.log2fc :List of 6
  ..$ : num  -2.62 1.58
  ..$ : num 1.86
  ..$ : num 5.31
  ..$ : num 1.25
  ..$ : num 1.34
  ..$ : num 1.35
 $ deg.p.adj  :List of 6
  ..$ : num  0.038544 0.000173
  ..$ : num 0.00187
  ..$ : num 0.00835
  ..$ : num 0.00291
  ..$ : num 0.0126
  ..$ : num 4.69e-06
```

3 Visualization of GO enrichment

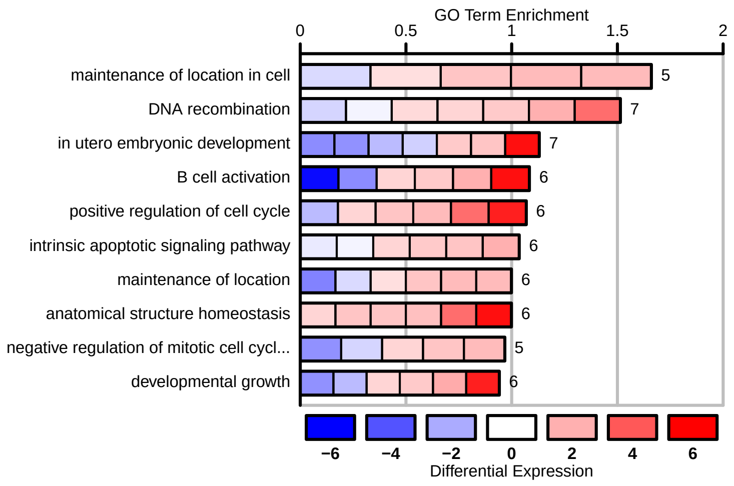
For the examples, we use a subset of the GO enrichment analysis of the golubstat data. Create the subset with:

```
> x <- subset(golubstat, p<=.05 & significant>4 & !duplicated(genes))
> x <- head(x, 10)
```

3.1 cell.plot

This function plots a horizontal barchart of strictly positive values in x. For each entry, a vector of additional values needs to be provided in a list. The additional values are plotted as cells subdividing the length of the corresponding bar. A typical usage scenario is plotting the enrichment of Gene Ontology terms, with individual cells reflecting the differential regulation of the constituent genes.

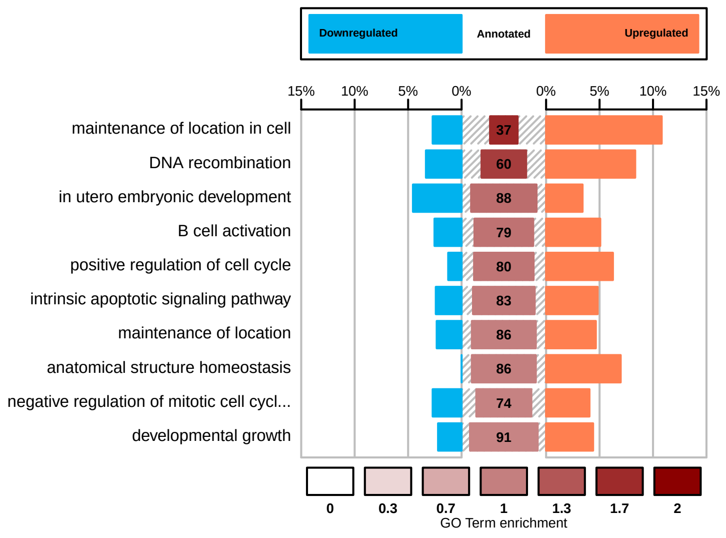
```
> cell.plot( x = setNames(x$loge, x$term), cells =x$deg.log2fc, x.bound = 2,
+            cell.bounds = c(-6,6), space = 0.15, main ="", y.mar =c(0.1,0),
+            x.mar =c(.47, 0), key.n=7, cex=1.6, grid.lwd=3, axis.cex=0.8, cell.outer=3 )
```



3.2 sym.plot

This function plots a split barchart, showing the proportions of two mutually exclusive sets in relation to a set containing them both. E.g., Gene Ontology terms, showing the proportions of differentially down-regulated and up-regulated annotated genes from a perturbation experiment. The color of the central column elements maps to the value provided in x (e.g. GO term enrichment). Associated genes may be provided as a list of vectors of expression values, same as for cell.plot(), or as separate vectors x.up and x.down, providing the numbers of up- and down-regulated genes in the same order as x.

```
> sym.plot( ticksize = 5, x = setNames(x$loge, x$term), cells = x$deg.log2fc,
+           x.annotated = x$annotated, y.mar = c(0.1,0), x.mar = c(.47, 0), key.n=7,
+           key.lab = "GO Term enrichment", cex = 1.6, axis.cex=0.8, group.cex=0.7,
+           grid.lwd=3 )
```



3.3 Visual parameters

Both functions provide a range of parameters to tweak the visual appearance of the plots.

- **Font sizes** of the various elements can be adjusted using the `cex` parameters.

- **Internal margins** are adjusted through `x.mar` and `y.mar`. Used in particular to accomodate the label text to the left-hand side.
- **Absolute scaling** can be achieved through the `bar.scale` parameter, which acts as a multiplier for a predetermined average bar height. This is meant to be used in conjunction with a graphics device of constant size, to ensure visual consistency among multiple plots that vary in the number of terms displayed.
- **Data ranges** for all output may be fixed:
  - `elem.bounds` Require the cardinality of a term or significant subset to be in a specific range, e.g. `c(10,100)` – between 10 and 100 genes.
  - `x.bound` Upper limit of the value displayed on the x-axis. For `sym.plot` this is must be a value between 0 and 100. The lower limit is always 0.
  - `cell.bounds` (only `cell.plot`) Sets the colour scale for cell data to a fixed range. If the plot contains values outside of that range, an indicator is added to the colour legend. This is particularly useful in the event of a few extreme outliers.
  - `mid.bounds` (only `sym.plot`) Similar to `cell.bounds`, this sets the colour range of the central column cells.
  - `sym` (only `cell.plot`), when set to `TRUE`, makes the scale of cell values symmetrical.
- **Colour schemes** may be controlled through character vectors provided to the following parameters. Internally, the R native function `ColorRampPalette()` is used.
  - `cell.col` (only `cell.plot`) Specify three valid colour names for cell data visualisation (lower extreme, mid point, upper extreme).
  - `mid.col` (only `sym.plot`) Specify two valid colour names to define the range of the central column cell colours.
- **Colour keys** can be adjusted in their resolution (number of boxes) using the `key.n` parameter.
- **Gridlines** may be hidden via the `gridlines` parameter.