# CellPlotManual

*Sven E. Templer, Robert Sehlke*

*2016-02-29*

## Contents



## Description

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

The development platform of this software is hosted on github. In case of any questions, consider using the issues system.

## Usage

### Load package and data

After installing the package (see Installation section), load it in a running **R** session:

```
library(CellPlot)
library(miscset) # only for examples in the vignette
```

The example data is available with the commands:

```
data("golubGO")
data("leukemiasGO")
```

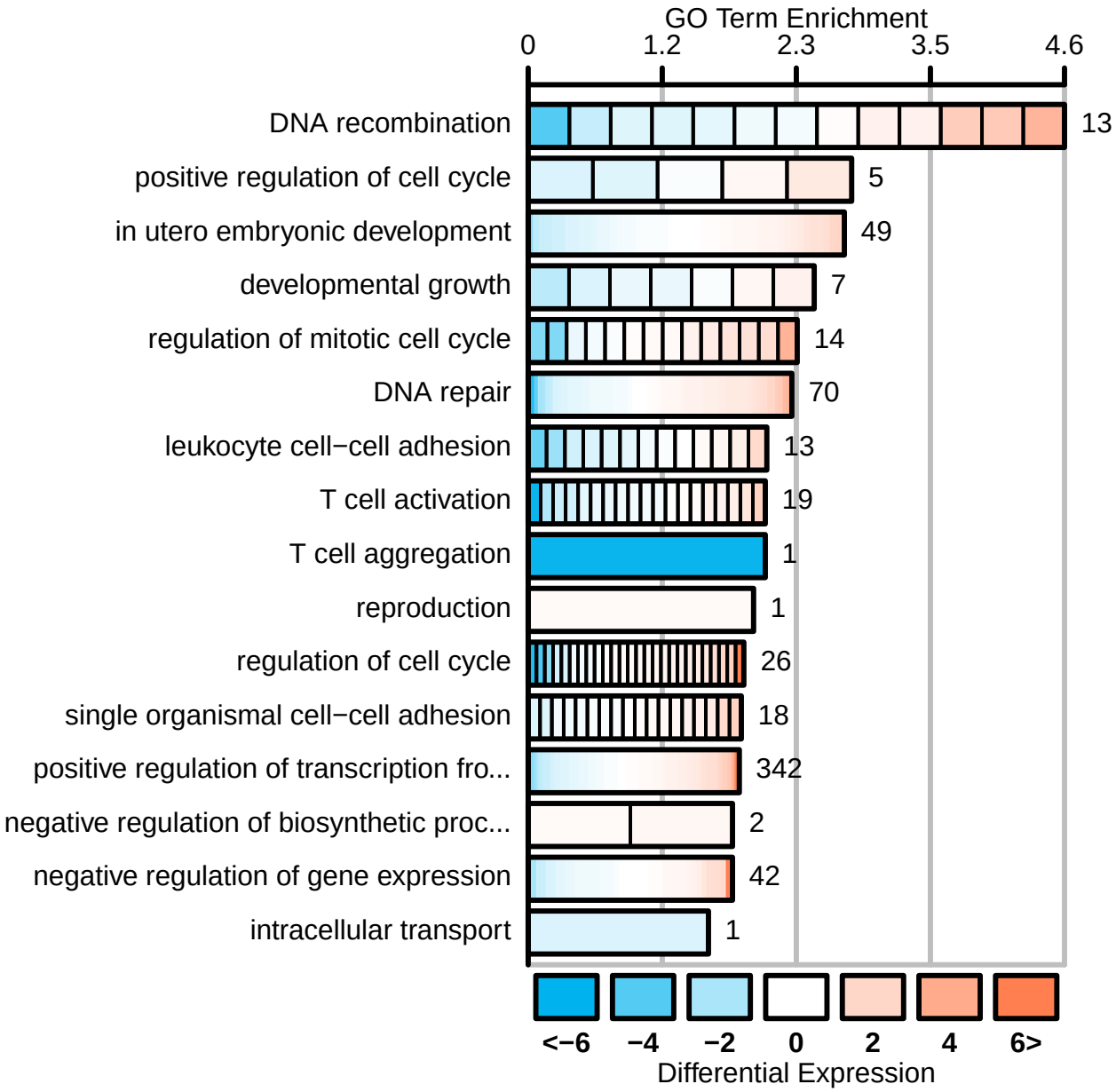
Reduce set of GO terms to display using the base **R** function `subset`:

```
x <- subset(golubGO$golub, pvalCutOff <= 0.05 & Significant > 5 & !duplicated(PROBEID))
x <- sort(x, decreasing = TRUE, by = "LogEnrich")
```

## Cell plot

The function `cell.plot` 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.

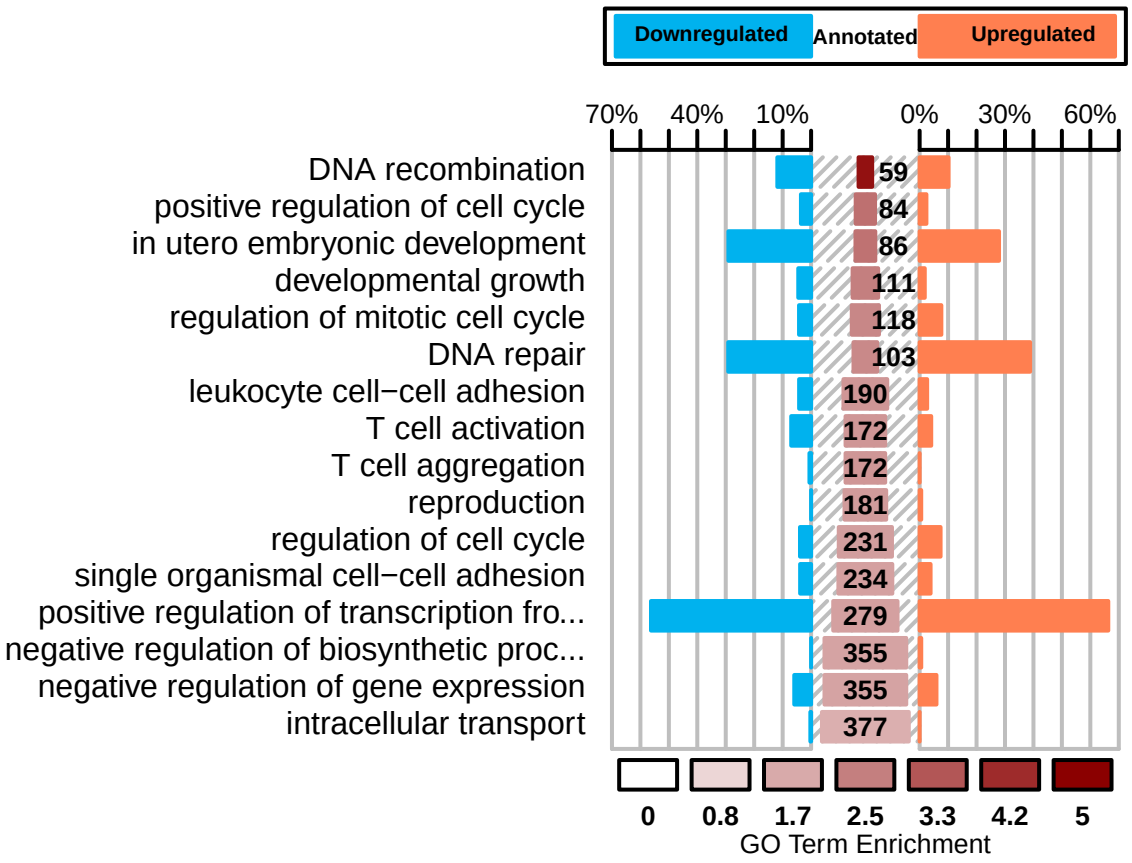
```
cell.plot(x = setNames(x$LogEnrich, x$Term), cells = x$log2fc,
          main ="GO enrichment on ALL/AML gene expression")
```




## Sym plot

The function `sym.plot` 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(x = setNames(x$LogEnrich, x$Term), cells = x$log2fc, x.annotated = x$Annotated,
         main ="GO enrichment on ALL/AML gene expression")
```




## Arc plot

```
x$deg.up <- lapply(Map(setNames, x$deg.log2fc, x$genes), function (i) { i[i>0] })
x$deg.down <- lapply(Map(setNames, x$deg.log2fc, x$genes), function (i) { i[i<0] })
arc.plot( x = setNames(x$loge, x$term), up.list = x$deg.up, down.list = x$deg.down, x.mar=c(1,0.5))
```

## 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.

# Workflows

Gene ontology term enrichment in differential gene expression data can be performed using the Bioconductor/topGO package.

## Dataset `golubGO`

This section provides code and comments on the workflow to perform differential expression of genes (DEG) analysis and the gene ontology term (GO) enrichment of from the results applied to the `golub` dataset from the Bioconductor/multtest package.

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 Bioconductor/annotate and Bioconductor/hu6800.db packages.

```
# CRAN
library(plyr)
# Bioconductor
library(topGO)
library(annotate)
library(hu6800.db)
library(multtest)

data(golub)

A <- split(golub, seq(nrow(golub)))
DEG <- do.call(rbind, mclapply(A, function(x){
```

```
  t <- t.test(x[as.logical(golub.cl)], x[!as.logical(golub.cl)])
  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$p.value,
    row.names = NULL)
}, mc.cores = 10))
DEG$padj <- p.adjust(DEG$p)
DEG <- data.frame(gene = golub.gnames[,3], DEG, stringsAsFactors = F)
DEG <- subset(DEG, !is.na(log2fc) & !is.na(padj))

M <- select(hu6800.db, DEG$gene, c("PROBEID","GO"))
genes <- dlply(subset(M, ONTOLOGY == "BP"), "PROBEID", function (x) unique(x$GO))
DEG <- subset(DEG, gene %in% names(genes))

GO <- new("topGOdata", ontology = "BP", description = 'golub',
          allGenes = setNames(DEG$padj, DEG$gene),
          geneSelectionFun = function (allScore) { allScore <= 0.05 },
          annotationFun = annFUN.gene2GO, gene2GO = genes)

GOsig <- lapply(list(golub=GO), function (x) {
  t <- new("elimCount", testStatistic = GOFisherTest, name = "Fisher test")
  s <- getSigGroups(x, t)
  r <- GenTable(x, pvalCutOff = s, topNodes = length(x@graph@nodes))
  r$LogEnrich <- r$Significant / r$Expected
  return(r)
})

golubGO <- Map(mergeGOdeg, GOsig, list(DEG), list(M), map.gene = "PROBEID", deg.p = "padj",deg.lfc="log
golubGO <- lapply(golubGO, subset, !is.na(PROBEID))
```

## Dataset `leukemiasGO`

This section provides code and comments on the workflow to perform differential expression of genes analysis and the gene ontology term enrichment of from the results applied to the `leukemiasEset` dataset from the Bioconductor/leukemiasEset package.

```
# CRAN
library(parallel)
library(plyr)
library(miscset)
# Bioconductor
library(DESeq2)
library(topGO)
library(annotate)
library(hu6800.db)
library(BiocParallel)
library(leukemiasEset)
```

```r
data(leukemiasEset)

M <- select(hu6800.db, featureNames(leukemiasEset), c("ENSEMBL","GO"), keytype = "ENSEMBL")
M <- subset(M, ONTOLOGY == "BP", c("ENSEMBL","GO"))
M <- M[!duplicated(M),]

genes <- dlply(M, "ENSEMBL", function (x) unique(x$GO))

A <- as(leukemiasEset,"data.frame")
A <- subset(A, LeukemiaType %in% c("NoL","ALL", "AML", "CLL"))
s <- subset(A, select = "LeukemiaType")
s$LeukemiaType <- relevel(factor(as.character(s$LeukemiaType)), "NoL")
A <- subset(A, select = colnames(A) %in% names(genes))
A <- sapply(A, as.integer)
A <- t(A)

DEG <- DESeqDataSetFromMatrix(countData = A, colData = s, design = ~ LeukemiaType)
# DEG <- estimateSizeFactors(DEG)
# DEG <- estimateDispersionsGeneEst(DEG)
# dispersions(DEG) <- mcols(DEG)$dispGeneEst
# DEG <- nbinomLRT(DEG, reduced = ~ 1)
DEG <- DESeq(DEG, fitType = "mean", parallel = T, BPPARAM = MulticoreParam(20))

DEG <- lapply(levels(s$LeukemiaType)[-1], function (x) {
  x <- results(DEG, c("LeukemiaType", x, "NoL"), "LeukemiaType", alpha = .05)
  x <- as.data.frame(x)
  x$gene <- rownames(x)
  return(x)
})
names(DEG) <- levels(s$LeukemiaType)[-1]

GO <- lapply(DEG, function (x) new(
  "topGOdata", ontology = "BP", description = 'Leukemia', allGenes = setNames(x$pvalue, x$gene),
  geneSelectionFun = function (allScore) { allScore <= 0.05 }, annotationFun = annFUN.gene2GO, gene2GO =

GOsig <- mclapply(GO, function (x) {
  t <- new("elimCount", testStatistic = GOFisherTest, name = "Fisher test")
  s <- getSigGroups(x, t)
  r <- GenTable(x, pvalCutOff = s, topNodes = length(x@graph@nodes))
  r$LogEnrich <- r$Significant / r$Expected
  return(r)
}, mc.cores = length(GO))

leukemiasGO <- Map(mergeGOdeg, GOsig, DEG, list(M), map.gene = "ENSEMBL")
leukemiasGO <- lapply(leukemiasGO, subset, !is.na(ENSEMBL))
leukemiasGO <- lapply(leukemiasGO, sort, TRUE, by = "LogEnrich")
```

# Installation

## CellPlot

Install the latest version from the repository on github.com/dieterich-lab/CellPlot. With the `devtools` package, it is an easy task:

```
install.packages("devtools")
library(devtools)
install_github('dieterich-lab/CellPlot', build_vignettes = TRUE)
```

## Dependencies

```
# load bioconductor functions
source("https://bioconductor.org/biocLite.R")

# install packages for CellPlot
install.packages("plyr")
biocLite("topGO")

# install packages for vignette, datasets, etc.
install.packages("miscset")
biocLite("leukemiasEset")
biocLite("multtest")
biocLite("hu6800.db")
```