RNA-Seq Workflow Template

Author: Daniela Cassol (danielac@ucr.edu) and Thomas Girke (thomas.girke@ucr.edu)

18 April 2020

Package

systemPipeR 1.21.6

Contents

1 Introduction

Users want to provide here background information about the design of their RNA-Seq project.

2 Samples and environment settings

2.1 Environment settings and input data

Typically, the user wants to record here the sources and versions of the reference genome sequence along with the corresponding annotations. In the provided sample data set all data inputs are stored in a `data` subdirectory and all results will be written to a separate `results` directory, while the `systemPipeRNAseq.Rmd` script and the `targets` file are expected to be located in the parent directory. The R session is expected to run from this parent directory.

To run this sample report, mini sample FASTQ and reference genome files can be downloaded from here. The chosen data set SRP010938 contains 18 paired-end (PE) read sets from *Arabidposis thaliana* (Howard et al. 2013). To minimize processing time during testing, each FASTQ file has been subsetted to 90,000-100,000 randomly sampled PE reads that map to the first 100,000 nucleotides of each chromosome of the *A. thalina* genome. The corresponding reference genome sequence (FASTA) and its GFF annotation files (provided in the same download) have been truncated accordingly. This way the entire test sample data set is less than 200MB in storage space. A PE read set has been chosen for this test data set for flexibility, because it can be used for testing both types of analysis routines requiring either SE (single end) reads or PE reads.

Instructions for loading the demo data available for NGS workflow templates into the user's current working directory can be found here. At the moment, the package includes workflow templates for RNA-Seq, ChIP-Seq, VAR-Seq and Ribo-Seq. Templates for additional NGS applications will be provided in the future.

Alternatively, this can be done from the command-line as find here.

Now open the R markdown script `systemPipeRNAseq.Rmd`in your R IDE (*e.g.* vim-r or RStudio) and run the workflow as outlined below. If you work under Vim-R-Tmux, the following command sequence will connect the user in an interactive session with a node on the cluster [TODO: fix, point to the main vignette]. The code of the `Rmd`script can then be sent from Vim on the login (head) node to an open R session running on the corresponding computer node. This is important since Tmux sessions should not be run on the computer nodes.

Now check whether your R session is running on a computer node of the cluster and not on a head node.

```
system("hostname")  # should return name of a compute node starting with i or c
getwd()  # checks current working directory of R session
dir()  # returns content of current working directory
```

2.2 Required packages and resources

The `systemPipeR` package needs to be loaded to perform the analysis steps shown in this report (H Backman and Girke 2016).

```
library(systemPipeR)
```

If applicable load custom functions not provided by `systemPipeR` package.

To apply workflows to custom data, the user needs to modify the *targets* file and if necessary update the corresponding *.cwl* and *.yml* files. A collection of pre-generated *.cwl* and *.yml* files are provided in the *param/cwl* subdirectory of each workflow template. They are also viewable in the GitHub repository of *systemPipeRdata* (see here).

2.3 Experiment definition provided by `targets` file

The `targets` file defines all FASTQ files and sample comparisons of the analysis workflow.

```
targetspath <- system.file("extdata", "targets.txt", package = "systemPipeR")
targets <- read.delim(targetspath, comment.char = "#")[, 1:4]
targets
##                      FileName SampleName Factor SampleLong
## 1  ./data/SRR446027_1.fastq.gz        M1A     M1  Mock.1h.A
## 2  ./data/SRR446028_1.fastq.gz        M1B     M1  Mock.1h.B
## 3  ./data/SRR446029_1.fastq.gz        A1A     A1   Avr.1h.A
## 4  ./data/SRR446030_1.fastq.gz        A1B     A1   Avr.1h.B
## 5  ./data/SRR446031_1.fastq.gz        V1A     V1   Vir.1h.A
## 6  ./data/SRR446032_1.fastq.gz        V1B     V1   Vir.1h.B
## 7  ./data/SRR446033_1.fastq.gz        M6A     M6  Mock.6h.A
## 8  ./data/SRR446034_1.fastq.gz        M6B     M6  Mock.6h.B
## 9  ./data/SRR446035_1.fastq.gz        A6A     A6   Avr.6h.A
## 10 ./data/SRR446036_1.fastq.gz        A6B     A6   Avr.6h.B
## 11 ./data/SRR446037_1.fastq.gz        V6A     V6   Vir.6h.A
## 12 ./data/SRR446038_1.fastq.gz        V6B     V6   Vir.6h.B
## 13 ./data/SRR446039_1.fastq.gz       M12A    M12 Mock.12h.A
## 14 ./data/SRR446040_1.fastq.gz       M12B    M12 Mock.12h.B
## 15 ./data/SRR446041_1.fastq.gz       A12A    A12  Avr.12h.A
## 16 ./data/SRR446042_1.fastq.gz       A12B    A12  Avr.12h.B
## 17 ./data/SRR446043_1.fastq.gz       V12A    V12  Vir.12h.A
## 18 ./data/SRR446044_1.fastq.gz       V12B    V12  Vir.12h.B
```

3 Read preprocessing

3.1 Read quality filtering and trimming

The function `preprocessReads` allows to apply predefined or custom read preprocessing functions to all FASTQ files referenced in a `SYSargs2` container, such as quality filtering or adapter trimming routines. The paths to the resulting output FASTQ files are stored in the `output` slot of the `SYSargs2` object. The following example performs adapter trimming with the `trimLRPatterns` function from the `Biostrings` package. After the trimming step a new targets file is generated (here `targets_trim.txt`) containing the paths to the trimmed FASTQ files. The new targets file can be used for the next workflow step with an updated `SYSargs2` instance, *e.g.* running the NGS alignments using the trimmed FASTQ files.

Construct *SYSargs2* object from *cwl* and *yml* param and *targets* files.

```
dir_path <- system.file("extdata/cwl/preprocessReads/trim-se",
    package = "systemPipeR")
trim <- loadWorkflow(targets = targetspath, wf_file = "trim-se.cwl",
    input_file = "trim-se.yml", dir_path = dir_path)
trim <- renderWF(trim, inputvars = c(FileName = "_FASTQ_PATH1_",
    SampleName = "_SampleName_"))
trim
output(trim)[1:2]
```

```
preprocessReads(args = trim, Fct = "trimLRPatterns(Rpattern='GCCCGGGTAA',
                subject=fq)",
    batchsize = 1e+05, overwrite = TRUE, compress = TRUE)
writeTargetsout(x = trim, file = "targets_trimPE.txt", step = 1,
    new_col = "FileName1", new_col_output_index = 1, overwrite = TRUE)
```

3.2 FASTQ quality report

The following `seeFastq` and `seeFastqPlot` functions generate and plot a series of useful quality statistics for a set of FASTQ files including per cycle quality box plots, base proportions, base-level quality trends, relative k-mer diversity, length and occurrence distribution of reads, number of reads above quality cutoffs and mean quality distribution. The results are written to a PDF file named `fastqReport.pdf`.

```
fqlist <- seeFastq(fastq = infile1(trim), batchsize = 10000,
    klength = 8)
pdf("./results/fastqReport.pdf", height = 18, width = 4 * length(fqlist))
seeFastqPlot(fqlist)
dev.off()
```

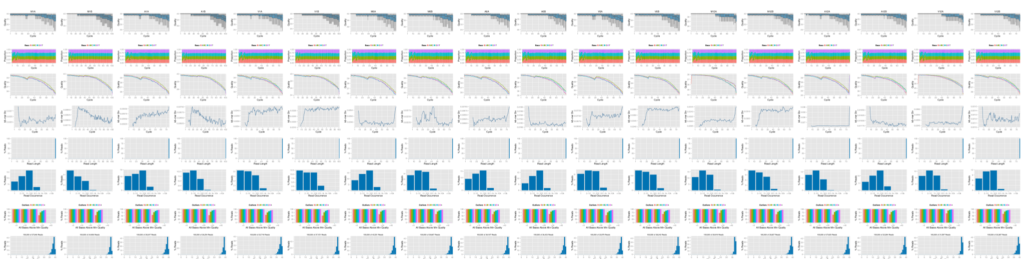

Figure 1: FASTQ quality report for 18 samples

4 Alignments

4.1 Read mapping with `HISAT2`

The following steps will demonstrate how to use the short read aligner `Hisat2` (Kim, Langmead, and Salzberg 2015) in both interactive job submissions and batch submissions to queuing systems of clusters using the `systemPipeR's` new CWL command-line interface.

Build `Hisat2` index.

```
dir_path <- system.file("extdata/cwl/hisat2/hisat2-idx", package = "systemPipeR")
idx <- loadWorkflow(targets = NULL, wf_file = "hisat2-index.cwl",
    input_file = "hisat2-index.yml", dir_path = dir_path)
idx <- renderWF(idx)
idx
cmdlist(idx)

## Run
runCommandline(idx, make_bam = FALSE)
```

The parameter settings of the aligner are defined in the `hisat2-mapping-se.cwl` and `hisat2-mapping-se.yml` files. The following shows how to construct the corresponding *SYSargs2* object, here *args*.

```
dir_path <- system.file("extdata/cwl/hisat2/hisat2-se", package = "systemPipeR")
args <- loadWorkflow(targets = targetspath, wf_file = "hisat2-mapping-se.cwl",
    input_file = "hisat2-mapping-se.yml", dir_path = dir_path)
args <- renderWF(args, inputvars = c(FileName = "_FASTQ_PATH1_",
    SampleName = "_SampleName_"))
args
cmdlist(args)[1:2]
output(args)[1:2]

## Runc single Machine
args <- runCommandline(args)
```

```
library(batchtools)
resources <- list(walltime = 120, ntasks = 1, ncpus = 4, memory = 1024)
reg <- clusterRun(args, FUN = runCommandline, more.args = list(args = args,
    make_bam = TRUE, dir = FALSE), conffile = ".batchtools.conf.R",
    template = "batchtools.slurm.tmpl", Njobs = 18, runid = "01",
    resourceList = resources)
getStatus(reg = reg)
waitForJobs(reg = reg)
args <- output_update(args, dir = FALSE, replace = TRUE, extension = c(".sam",
    ".bam"))  ## Updates the output(args) to the right location in the subfolders
output(args)
```

Check whether all BAM files have been created.

```
outpaths <- subsetWF(args, slot = "output", subset = 1, index = 1)
file.exists(outpaths)
```

4.2 Read and alignment stats

The following provides an overview of the number of reads in each sample and how many of them aligned to the reference.

```
read_statsDF <- alignStats(args = args)
write.table(read_statsDF, "results/alignStats.xls", row.names = FALSE,
    quote = FALSE, sep = "\t")
```

The following shows the alignment statistics for a sample file provided by the `systemPipeR` package.

```
read.table(system.file("extdata", "alignStats.xls", package = "systemPipeR"),
    header = TRUE)[1:4, ]
##   FileName Nreads2x Nalign Perc_Aligned Nalign_Primary
## 1      M1A   192918 177961     92.24697         177961
## 2      M1B   197484 159378     80.70426         159378
## 3      A1A   189870 176055     92.72397         176055
## 4      A1B   188854 147768     78.24457         147768
##   Perc_Aligned_Primary
## 1             92.24697
## 2             80.70426
## 3             92.72397
## 4             78.24457
```

4.3 Create symbolic links for viewing BAM files in IGV

The `symLink2bam` function creates symbolic links to view the BAM alignment files in a genome browser such as IGV. The corresponding URLs are written to a file with a path specified under `urlfile` in the `results` directory.

```
symLink2bam(sysargs = args, htmldir = c("~/.html/", "somedir/"),
    urlbase = "http://biocluster.ucr.edu/~tgirke/", urlfile = "./results/IGVurl.txt")
```

5 Read quantification

5.1 Read counting with `summarizeOverlaps` in parallel mode using multiple cores

Reads overlapping with annotation ranges of interest are counted for each sample using the `summarizeOverlaps` function (Lawrence et al. 2013). The read counting is preformed for exonic gene regions in a non-strand-specific manner while ignoring overlaps among different genes. Subsequently, the expression count values are normalized by *reads per kp per million mapped reads* (RPKM). The raw read count table (`countDFeByg.xls`) and the corresponding RPKM table (`rpkmDFeByg.xls`) are written to separate files in the directory of this project. Parallelization is achieved with the `BiocParallel` package, here using 8 CPU cores.

```
library("GenomicFeatures")
library(BiocParallel)
txdb <- makeTxDbFromGFF(file = "data/tair10.gff", format = "gff",
    dataSource = "TAIR", organism = "Arabidopsis thaliana")
saveDb(txdb, file = "./data/tair10.sqlite")
txdb <- loadDb("./data/tair10.sqlite")
outpaths <- subsetWF(args, slot = "output", subset = 1, index = 1)
(align <- readGAlignments(outpaths[1]))  # Demonstrates how to read bam file into R
eByg <- exonsBy(txdb, by = c("gene"))
bfl <- BamFileList(outpaths, yieldSize = 50000, index = character())
multicoreParam <- MulticoreParam(workers = 2)
register(multicoreParam)
```

```
registered()
counteByg <- bplapply(bfl, function(x) summarizeOverlaps(eByg,
    x, mode = "Union", ignore.strand = TRUE, inter.feature = FALSE,
    singleEnd = TRUE))
countDFeByg <- sapply(seq(along = counteByg), function(x) assays(counteByg[[x]])$counts)
rownames(countDFeByg) <- names(rowRanges(counteByg[[1]]))
colnames(countDFeByg) <- names(bfl)
rpkmDFeByg <- apply(countDFeByg, 2, function(x) returnRPKM(counts = x,
    ranges = eByg))
write.table(countDFeByg, "results/countDFeByg.xls", col.names = NA,
    quote = FALSE, sep = "\t")
write.table(rpkmDFeByg, "results/rpkmDFeByg.xls", col.names = NA,
    quote = FALSE, sep = "\t")
```

Sample of data slice of count table

```
read.delim("results/countDFeByg.xls", row.names = 1, check.names = FALSE)[1:4,
    1:5]
```

Sample of data slice of RPKM table

```
read.delim("results/rpkmDFeByg.xls", row.names = 1, check.names = FALSE)[1:4,
    1:4]
```

Note, for most statistical differential expression or abundance analysis methods, such as `edgeR` or `DESeq2`, the raw count values should be used as input. The usage of RPKM values should be restricted to specialty applications required by some users, *e.g.* manually comparing the expression levels among different genes or features.

5.2 Sample-wise correlation analysis

The following computes the sample-wise Spearman correlation coefficients from the `rlog` transformed expression values generated with the `DESeq2` package. After transformation to a distance matrix, hierarchical clustering is performed with the `hclust` function and the result is plotted as a dendrogram (also see file `sample_tree.pdf`).

```
library(DESeq2, quietly = TRUE)
library(ape, warn.conflicts = FALSE)
countDF <- as.matrix(read.table("./results/countDFeByg.xls"))
colData <- data.frame(row.names = targets.as.df(targets(args))$SampleName,
    condition = targets.as.df(targets(args))$Factor)
dds <- DESeqDataSetFromMatrix(countData = countDF, colData = colData,
    design = ~condition)
d <- cor(assay(rlog(dds)), method = "spearman")
hc <- hclust(dist(1 - d))
pdf("results/sample_tree.pdf")
plot.phylo(as.phylo(hc), type = "p", edge.col = "blue", edge.width = 2,
    show.node.label = TRUE, no.margin = TRUE)
dev.off()
```

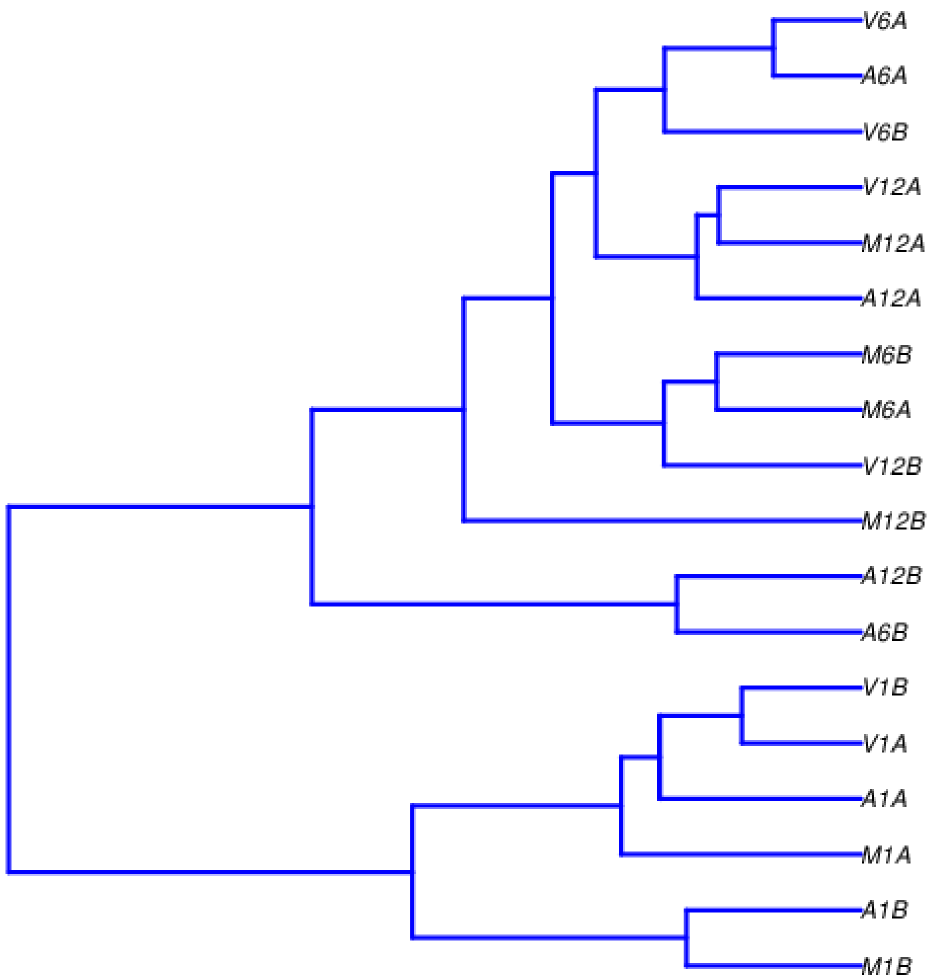

Figure 2: Correlation dendrogram of samples

6 Analysis of DEGs

The analysis of differentially expressed genes (DEGs) is performed with the glm method of the `edgeR` package (Robinson, McCarthy, and Smyth 2010). The sample comparisons used by this analysis are defined in the header lines of the `targets.txt` file starting with `<CMP>`.

6.1 Run edgeR

```
library(edgeR)
countDF <- read.delim("results/countDFeByg.xls", row.names = 1,
    check.names = FALSE)
targets <- read.delim("targets.txt", comment = "#")
cmp <- readComp(file = "targets.txt", format = "matrix", delim = "-")
edgeDF <- run_edgeR(countDF = countDF, targets = targets, cmp = cmp[[1]],
    independent = FALSE, mdsplot = "")
```

Add gene descriptions

```
library("biomaRt")
m <- useMart("plants_mart", dataset = "athaliana_eg_gene", host = "plants.ensembl.org")
desc <- getBM(attributes = c("tair_locus", "description"), mart = m)
desc <- desc[!duplicated(desc[, 1]), ]
descv <- as.character(desc[, 2])
names(descv) <- as.character(desc[, 1])
edgeDF <- data.frame(edgeDF, Desc = descv[rownames(edgeDF)],
    check.names = FALSE)
write.table(edgeDF, "./results/edgeRglm_allcomp.xls", quote = FALSE,
    sep = "\t", col.names = NA)
```

6.2 Plot DEG results

Filter and plot DEG results for up and down regulated genes. The definition of *up* and *down* is given in the corresponding help file. To open it, type `?filterDEGs` in the R console.

```
edgeDF <- read.delim("results/edgeRglm_allcomp.xls", row.names = 1,
    check.names = FALSE)
pdf("results/DEGcounts.pdf")
DEG_list <- filterDEGs(degDF = edgeDF, filter = c(Fold = 2, FDR = 20))
dev.off()
write.table(DEG_list$Summary, "./results/DEGcounts.xls", quote = FALSE,
    sep = "\t", row.names = FALSE)
```

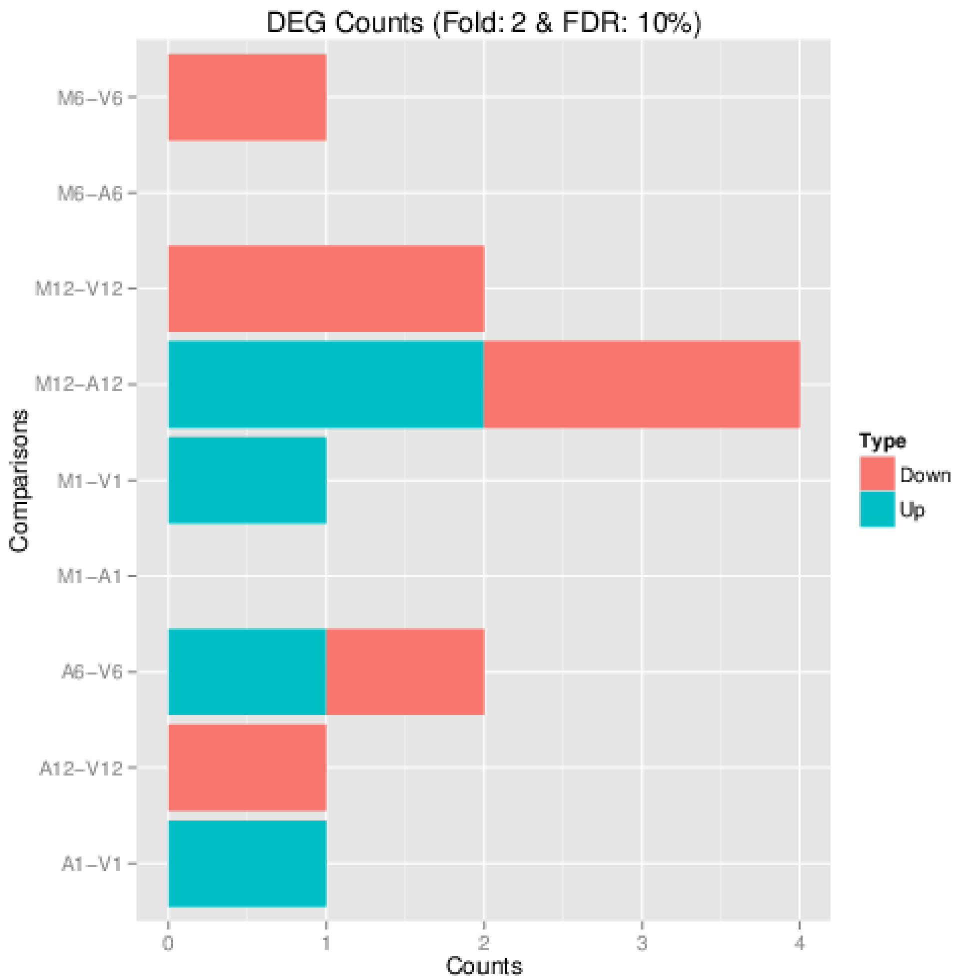

Figure 3: Up and down regulated DEGs with FDR of 1%

6.3 Venn diagrams of DEG sets

The `overLapper` function can compute Venn intersects for large numbers of sample sets (up to 20 or more) and plots 2-5 way Venn diagrams. A useful feature is the possibility to combine the counts from several Venn comparisons with the same number of sample sets in a single Venn diagram (here for 4 up and down DEG sets).

```
vennsetup <- overLapper(DEG_list$Up[6:9], type = "vennsets")
vennsetdown <- overLapper(DEG_list$Down[6:9], type = "vennsets")
pdf("results/vennplot.pdf")
vennPlot(list(vennsetup, vennsetdown), mymain = "", mysub = "",
    colmode = 2, ccol = c("blue", "red"))
dev.off()
```

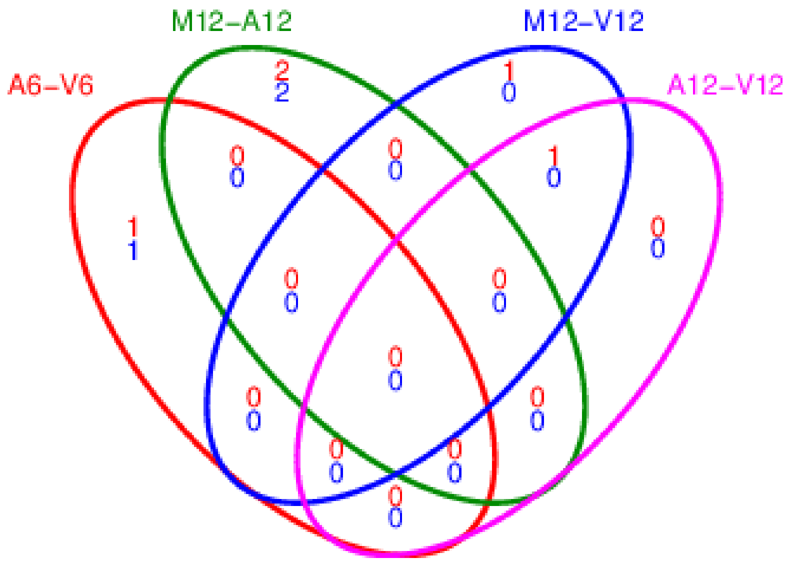

Figure 4: Venn Diagram for 4 Up and Down DEG Sets

7 GO term enrichment analysis

7.1 Obtain gene-to-GO mappings

The following shows how to obtain gene-to-GO mappings from `biomaRt` (here for *A. thaliana*) and how to organize them for the downstream GO term enrichment analysis. Alternatively, the gene-to-GO mappings can be obtained for many organisms from Bioconductor's `*.db` genome annotation packages or GO annotation files provided by various genome databases. For each annotation this relatively slow preprocessing step needs to be performed only once. Subsequently, the preprocessed data can be loaded with the `load` function as shown in the next subsection.

```
library("biomaRt")
listMarts()  # To choose BioMart database
listMarts(host = "plants.ensembl.org")
m <- useMart("plants_mart", host = "plants.ensembl.org")
```

```
listDatasets(m)
m <- useMart("plants_mart", dataset = "athaliana_eg_gene", host = "plants.ensembl.org")
listAttributes(m)  # Choose data types you want to download
go <- getBM(attributes = c("go_id", "tair_locus", "namespace_1003"),
    mart = m)
go <- go[go[, 3] != "", ]
go[, 3] <- as.character(go[, 3])
go[go[, 3] == "molecular_function", 3] <- "F"
go[go[, 3] == "biological_process", 3] <- "P"
go[go[, 3] == "cellular_component", 3] <- "C"
go[1:4, ]
dir.create("./data/GO")
write.table(go, "data/GO/GOannotationsBiomart_mod.txt", quote = FALSE,
    row.names = FALSE, col.names = FALSE, sep = "\t")
catdb <- makeCATdb(myfile = "data/GO/GOannotationsBiomart_mod.txt",
    lib = NULL, org = "", colno = c(1, 2, 3), idconv = NULL)
save(catdb, file = "data/GO/catdb.RData")
```

7.2 Batch GO term enrichment analysis

Apply the enrichment analysis to the DEG sets obtained the above differential expression analysis. Note, in the following example the `FDR` filter is set here to an unreasonably high value, simply because of the small size of the toy data set used in this vignette. Batch enrichment analysis of many gene sets is performed with the function. When `method=all`, it returns all GO terms passing the p-value cutoff specified under the `cutoff` arguments. When `method=slim`, it returns only the GO terms specified under the `myslimv` argument. The given example shows how a GO slim vector for a specific organism can be obtained from BioMart.

```
library("biomaRt")
load("data/GO/catdb.RData")
DEG_list <- filterDEGs(degDF = edgeDF, filter = c(Fold = 2, FDR = 50),
    plot = FALSE)
up_down <- DEG_list$UporDown
names(up_down) <- paste(names(up_down), "_up_down", sep = "")
up <- DEG_list$Up
names(up) <- paste(names(up), "_up", sep = "")
down <- DEG_list$Down
names(down) <- paste(names(down), "_down", sep = "")
DEGlist <- c(up_down, up, down)
DEGlist <- DEGlist[sapply(DEGlist, length) > 0]
BatchResult <- GOCluster_Report(catdb = catdb, setlist = DEGlist,
    method = "all", id_type = "gene", CLSZ = 2, cutoff = 0.9,
    gocats = c("MF", "BP", "CC"), recordSpecGO = NULL)
library("biomaRt")
m <- useMart("plants_mart", dataset = "athaliana_eg_gene", host = "plants.ensembl.org")
goslimvec <- as.character(getBM(attributes = c("goslim_goa_accession"),
    mart = m)[, 1])
BatchResultslim <- GOCluster_Report(catdb = catdb, setlist = DEGlist,
    method = "slim", id_type = "gene", myslimv = goslimvec, CLSZ = 10,
    cutoff = 0.01, gocats = c("MF", "BP", "CC"), recordSpecGO = NULL)
```

7.3 Plot batch GO term results

The `data.frame` generated by `GOCluster` can be plotted with the `goBarplot` function. Because of the variable size of the sample sets, it may not always be desirable to show the results from different DEG sets in the same bar plot. Plotting single sample sets is achieved by subsetting the input data frame as shown in the first line of the following example.

```
gos <- BatchResultslim[grep("M6-V6_up_down", BatchResultslim$CLID),
    ]
gos <- BatchResultslim
pdf("GOslimbarplotMF.pdf", height = 8, width = 10)
goBarplot(gos, gocat = "MF")
dev.off()
goBarplot(gos, gocat = "BP")
goBarplot(gos, gocat = "CC")
```

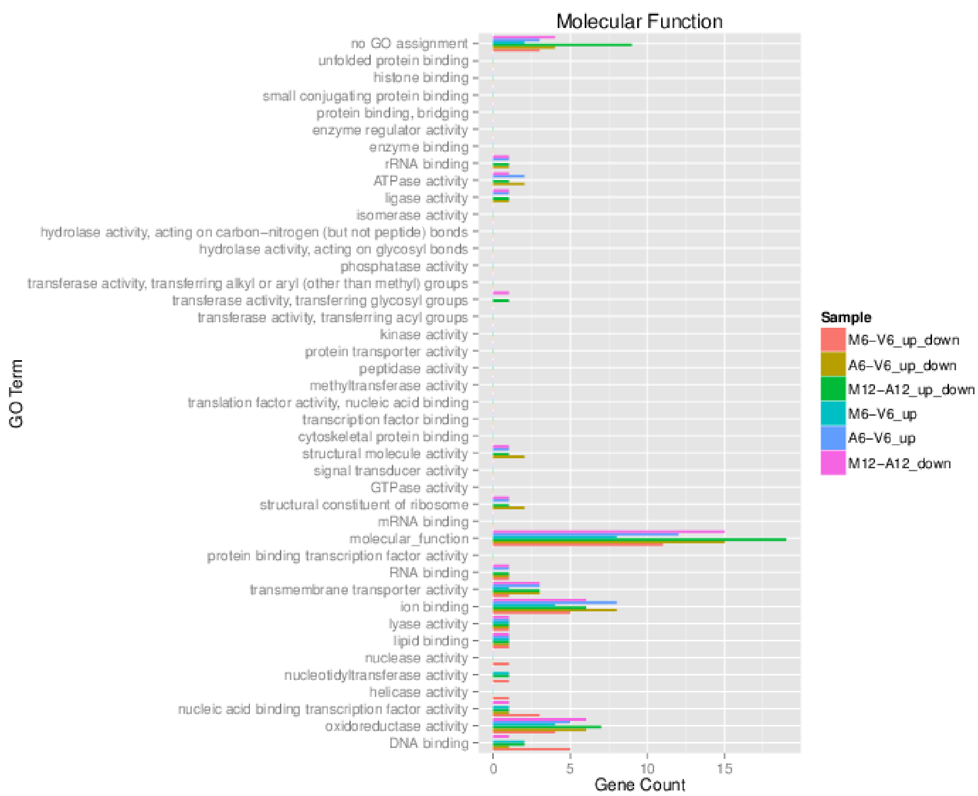

Figure 5: GO Slim Barplot for MF Ontology

8 Clustering and heat maps

The following example performs hierarchical clustering on the `rlog` transformed expression matrix subsetted by the DEGs identified in the above differential expression analysis. It uses a Pearson correlation-based distance measure and complete linkage for cluster joining.

```
library(pheatmap)
geneids <- unique(as.character(unlist(DEG_list[[1]])))
y <- assay(rlog(dds))[geneids, ]
pdf("heatmap1.pdf")
pheatmap(y, scale = "row", clustering_distance_rows = "correlation",
    clustering_distance_cols = "correlation")
dev.off()
```

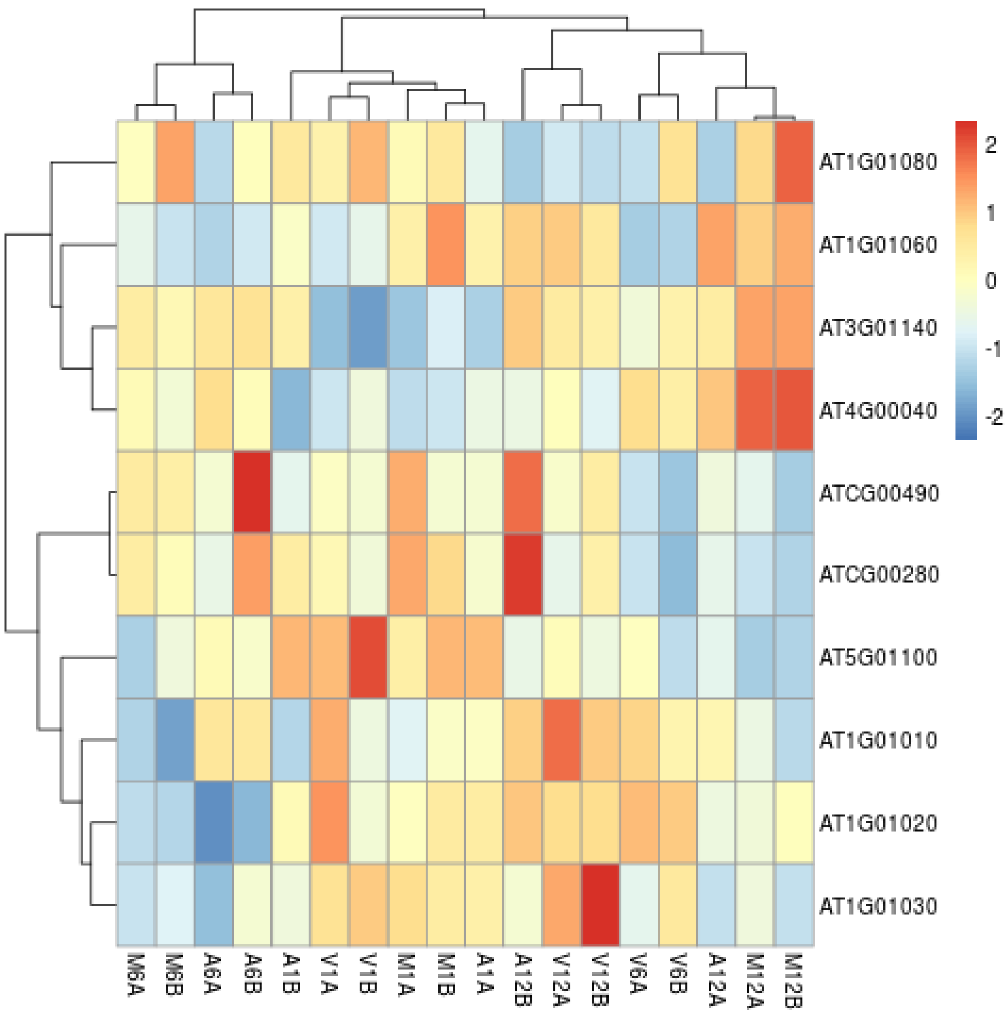

Figure 6: Heat Map with Hierarchical Clustering Dendrograms of DEGs

9 Version Information

```
sessionInfo()
## R Under development (unstable) (2020-03-31 r78116)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 20.04 LTS
##
```

```
## Matrix products: default
## BLAS:   /usr/local/lib/R/lib/libRblas.so
## LAPACK: /usr/local/lib/R/lib/libRlapack.so
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats4    parallel  stats     graphics  grDevices
## [6] utils     datasets  methods   base
##
## other attached packages:
##  [1] batchtools_0.9.13           data.table_1.12.8
##  [3] ape_5.3                     ggplot2_3.3.0
##  [5] systemPipeR_1.21.6          ShortRead_1.45.4
##  [7] GenomicAlignments_1.23.2    SummarizedExperiment_1.17.5
##  [9] DelayedArray_0.13.12        matrixStats_0.56.0
## [11] Biobase_2.47.3              BiocParallel_1.21.2
## [13] Rsamtools_2.3.7             Biostrings_2.55.7
## [15] XVector_0.27.2              GenomicRanges_1.39.3
## [17] GenomeInfoDb_1.23.16        IRanges_2.21.8
## [19] S4Vectors_0.25.15           BiocGenerics_0.33.3
## [21] BiocStyle_2.15.6
##
## loaded via a namespace (and not attached):
##  [1] colorspace_1.4-1         rjson_0.2.20
##  [3] hwriter_1.3.2            ellipsis_0.3.0
##  [5] bit64_0.9-7              AnnotationDbi_1.49.1
##  [7] fansi_0.4.1              codetools_0.2-16
##  [9] splines_4.1.0            knitr_1.28
## [11] annotate_1.65.1          GO.db_3.10.0
## [13] dbplyr_1.4.2             png_0.1-7
## [15] pheatmap_1.0.12          graph_1.65.2
## [17] BiocManager_1.30.10      compiler_4.1.0
## [19] httr_1.4.1               GOstats_2.53.0
## [21] backports_1.1.6          assertthat_0.2.1
## [23] Matrix_1.2-18            limma_3.43.6
## [25] cli_2.0.2                formatR_1.7
## [27] htmltools_0.4.0          prettyunits_1.1.1
## [29] tools_4.1.0              gtable_0.3.0
## [31] glue_1.4.0               GenomeInfoDbData_1.2.2
## [33] Category_2.53.1          dplyr_0.8.5
## [35] rappdirs_0.3.1           Rcpp_1.0.4.6
## [37] vctrs_0.2.4              nlme_3.1-145
## [39] rtracklayer_1.47.0       xfun_0.13
## [41] stringr_1.4.0            lifecycle_0.2.0
```

```
## [43] XML_3.99-0.3                edgeR_3.29.1
## [45] zlibbioc_1.33.1             scales_1.1.0
## [47] BSgenome_1.55.4             VariantAnnotation_1.33.4
## [49] hms_0.5.3                   RBGL_1.63.1
## [51] RColorBrewer_1.1-2          yaml_2.2.1
## [53] curl_4.3                    memoise_1.1.0
## [55] biomaRt_2.43.5              latticeExtra_0.6-29
## [57] stringi_1.4.6               RSQLite_2.2.0
## [59] genefilter_1.69.0           checkmate_2.0.0
## [61] GenomicFeatures_1.39.7      rlang_0.4.5
## [63] pkgconfig_2.0.3             bitops_1.0-6
## [65] evaluate_0.14               lattice_0.20-40
## [67] purrr_0.3.3                 bit_1.1-15.2
## [69] tidyselect_1.0.0            GSEABase_1.49.1
## [71] AnnotationForge_1.29.2      magrittr_1.5
## [73] bookdown_0.18               R6_2.4.1
## [75] base64url_1.4               DBI_1.1.0
## [77] pillar_1.4.3                withr_2.1.2
## [79] survival_3.1-11             RCurl_1.98-1.1
## [81] tibble_3.0.0                crayon_1.3.4
## [83] BiocFileCache_1.11.5        rmarkdown_2.1
## [85] jpeg_0.1-8.1                progress_1.2.2
## [87] locfit_1.5-9.4              grid_4.1.0
## [89] blob_1.2.1                  Rgraphviz_2.31.0
## [91] digest_0.6.25               xtable_1.8-4
## [93] brew_1.0-6                  openssl_1.4.1
## [95] munsell_0.5.0               askpass_1.1
```

10 Funding

This project was supported by funds from the National Institutes of Health (NIH).

References

H Backman, Tyler W, and Thomas Girke. 2016. "systemPipeR: NGS workflow and report generation environment." *BMC Bioinformatics* 17 (1): 388. https://doi.org/10.1186/s12859-016-1241-0.

Howard, Brian E, Qiwen Hu, Ahmet Can Babaoglu, Manan Chandra, Monica Borghi, Xiaoping Tan, Luyan He, et al. 2013. "High-Throughput RNA Sequencing of Pseudomonas-Infected Arabidopsis Reveals Hidden Transcriptome Complexity and Novel Splice Variants." *PLoS One* 8 (10): e74183. https://doi.org/10.1371/journal.pone.0074183.

Kim, Daehwan, Ben Langmead, and Steven L Salzberg. 2015. "HISAT: A Fast Spliced Aligner with Low Memory Requirements." *Nat. Methods* 12 (4): 357–60.

Lawrence, Michael, Wolfgang Huber, Hervé Pagès, Patrick Aboyoun, Marc Carlson, Robert Gentleman, Martin T Morgan, and Vincent J Carey. 2013. "Software for Computing and Annotating Genomic Ranges." *PLoS Comput. Biol.* 9 (8): e1003118. https://doi.org/10.1371/journal.pcbi.1003118.

Robinson, M D, D J McCarthy, and G K Smyth. 2010. "EdgeR: A Bioconductor Package for Differential Expression Analysis of Digital Gene Expression Data." *Bioinformatics* 26 (1): 139–40. https://doi.org/10.1093/bioinformatics/btp616.