Ribo-Seq workflow template: Some Descriptive Title

Project ID: RIBOseq_PI_Name_Organism_Aug2015
Project PI: First Last (first.last@inst.edu)
Author of Report: First Last (first.last@inst.edu)

October 5, 2015

Contents

1 Introduction

Note: this workflow is currently under construction!

1.1 Background and objectives

This workflow contains most of the data analysis steps described by Juntawong et al. (2014). It includes functionalities for processing both polyRibo-Seq and Ribo-Seq experiments. The former employs RNA bound to ribosomes and the latter uses the RNA fragments obtained from ribosome footprinting experiments. To improve re-usability and adapt to recent versions of software (*e.g.* R, Bioconductor and short read aligners), the code has been optimized accordingly. Thus, the results obtained with the updated code are expected to be similar but not necessarily identical with the published results described in the original paper.

1.2 Experimental design

Typically, users want to specify here all information relevant for the analysis of their NGS study. This includes detailed descriptions of FASTQ files, experimental design, reference genome, gene annotations, etc.

2 Load workflow environment

2.1 Load packages and sample data

The *systemPipeR* package needs to be loaded to perform the analysis steps shown in this report (Girke, 2014).

```
library(systemPipeR)
```

Load workflow environment with sample data into your current working directory. The sample data are described here.

```
library(systemPipeRdata)
genWorkenvir(workflow="ribseq")
setwd("riboseq")
```

In the workflow environments generated by `genWorkenvir` all data inputs are stored in a `data/` directory and all analysis results will be written to a separate `results/` directory, while the `systemPipeRIBOseq.Rnw` 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. Additional parameter files are stored under `param/`.

To work with real data, users want to organize their own data similarly and substitute all test data for their own data. To rerun an established workflow on new data, the initial `targets` file along with the corresponding FASTQ files are usually the only inputs the user needs to provide.

If applicable users can load custom functions not provided by *systemPipeR*. Skip this step if this is not the case.

```
source("systemPipeRIBOseq_Fct.R")
```

2.2 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        M1A     M1  Mock.1h.A
2  ./data/SRR446028_1.fastq        M1B     M1  Mock.1h.B
3  ./data/SRR446029_1.fastq        A1A     A1   Avr.1h.A
4  ./data/SRR446030_1.fastq        A1B     A1   Avr.1h.B
5  ./data/SRR446031_1.fastq        V1A     V1   Vir.1h.A
6  ./data/SRR446032_1.fastq        V1B     V1   Vir.1h.B
7  ./data/SRR446033_1.fastq        M6A     M6  Mock.6h.A
8  ./data/SRR446034_1.fastq        M6B     M6  Mock.6h.B
9  ./data/SRR446035_1.fastq        A6A     A6   Avr.6h.A
10 ./data/SRR446036_1.fastq        A6B     A6   Avr.6h.B
11 ./data/SRR446037_1.fastq        V6A     V6   Vir.6h.A
12 ./data/SRR446038_1.fastq        V6B     V6   Vir.6h.B
13 ./data/SRR446039_1.fastq       M12A    M12 Mock.12h.A
14 ./data/SRR446040_1.fastq       M12B    M12 Mock.12h.B
15 ./data/SRR446041_1.fastq       A12A    A12  Avr.12h.A
16 ./data/SRR446042_1.fastq       A12B    A12  Avr.12h.B
17 ./data/SRR446043_1.fastq       V12A    V12  Vir.12h.A
18 ./data/SRR446044_1.fastq       V12B    V12  Vir.12h.B
```

3 Read preprocessing

3.1 Quality filtering and adaptor trimming

The following custom function trims adaptors hierarchically from the longest to the shortest match of the right end of the reads. If `internalmatch=TRUE` then internal matches will trigger the same behavior. The argument `minpatternlength` defines the shortest adaptor match to consider in this iterative process. In addition, the function removes reads containing Ns or homopolymer regions. More detailed information on read preprocessing is provided in *systemPipeR*'s main vignette.

```
args <- systemArgs(sysma="param/trim.param", mytargets="targets.txt")
fctpath <- system.file("extdata", "custom_Fct.R", package="systemPipeR")
source(fctpath)
iterTrim <- ".iterTrimbatch1(fq, pattern='ACACGTCT', internalmatch=FALSE, minpatternlength=6, Nnumber=1, p
preprocessReads(args=args, Fct=iterTrim, batchsize=100000, overwrite=TRUE, compress=TRUE)
writeTargetsout(x=args, file="targets_trim.txt", 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`.

```
args <- systemArgs(sysma="param/tophat.param", mytargets="targets_trim.txt")
fqlist <- seeFastq(fastq=infile1(args), batchsize=100000, klength=8)
pdf("./results/fastqReport.pdf", height=18, width=4*length(fqlist))
seeFastqPlot(fqlist)
dev.off()
```

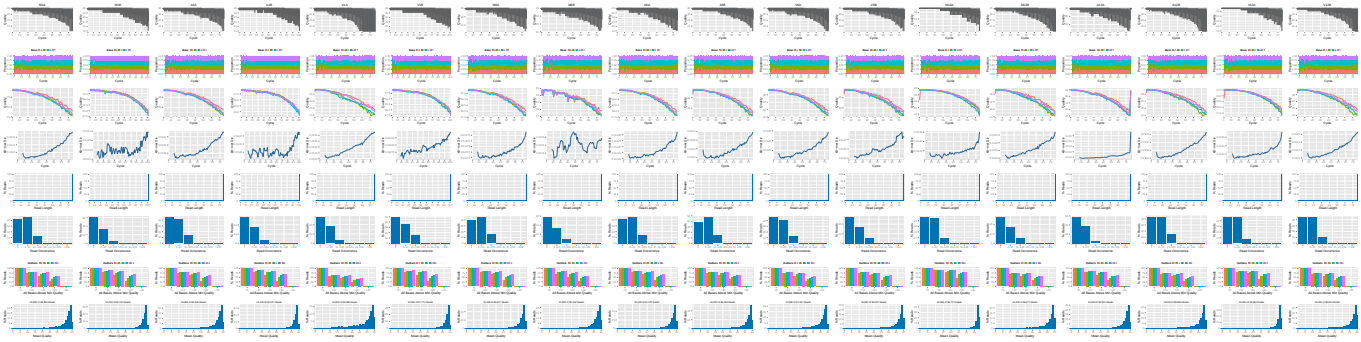

Figure 1: QC report for 18 FASTQ files.

4 Alignments

4.1 Read mapping with `Bowtie2/Tophat2`

The NGS reads of this project will be aligned against the reference genome sequence using `Bowtie2/TopHat2` (Kim et al., 2013; Langmead and Salzberg, 2012). The parameter settings of the aligner are defined in the `tophat.param` file.

```
args <- systemArgs(sysma="param/tophat.param", mytargets="targets.txt")
sysargs(args)[1] # Command-line parameters for first FASTQ file
```

Submission of alignment jobs to compute cluster, here using 72 CPU cores (18 `qsub` processes each with 4 CPU cores).

```
moduleload(modules(args))
system("bowtie2-build ./data/tair10.fasta ./data/tair10.fasta")
resources <- list(walltime="20:00:00", nodes=paste0("1:ppn=", cores(args)), memory="10gb")
reg <- clusterRun(args, conffile=".BatchJobs.R", template="torque.tmpl", Njobs=18, runid="01",
                  resourceList=resources)
waitForJobs(reg)
```

Check whether all BAM files have been created

```
file.exists(outpaths(args))
```

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")
```

```
read.table(system.file("extdata", "alignStats.xls", package="systemPipeR"), header=TRUE)[1:4,]
```

```
  FileName Nreads2x Nalign Perc_Aligned Nalign_Primary Perc_Aligned_Primary
1      M1A   192918 177961     92.24697         177961             92.24697
2      M1B   197484 159378     80.70426         159378             80.70426
3      A1A   189870 176055     92.72397         176055             92.72397
4      A1B   188854 147768     78.24457         147768             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`, here IGVurl.txt.

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

5 Read distribution across genomic features

The `genFeatures` function generates a variety of feature types from `TxDb` objects using utilities provided by the *GenomicFeatures* package.

5.1 Obtain feature types

The first step is the generation of the feature type ranges based on annotations provided by a GFF file that can be transformed into a `TxDb` object. This includes ranges for mRNAs, exons, introns, UTRs, CDSs, miRNAs, rRNAs, tRNAs, promoter and intergenic regions. In addition, any number of custom annotations can be included in this routine.

```
library(GenomicFeatures)
file <- system.file("extdata/annotation", "tair10.gff", package="systemPipeRdata")
txdb <- makeTxDbFromGFF(file=file, format="gff3", organism="Arabidopsis")
feat <- genFeatures(txdb, featuretype="all", reduce_ranges=TRUE, upstream=1000, downstream=0, verbose=TRUE
```

5.2 Count and plot reads of any length

The `featuretypeCounts` function counts how many reads in short read alignment files (BAM format) overlap with entire annotation categories. This utility is useful for analyzing the distribution of the read mappings across feature types, *e.g.* coding versus non-coding genes. By default the read counts are reported for the sense and antisense strand of each feature type separately. To minimize memory consumption, the BAM files are processed in a stream using utilities from the *Rsamtools* and *GenomicAlignment* packages. The counts can be reported for each read length separately or as a single value for reads of any length. Subsequently, the counting results can be plotted with the associated `plotfeaturetypeCounts` function.

The following generates and plots feature counts for any read length.

```
fc <- featuretypeCounts(bfl=BamFileList(outpaths(args), yieldSize=50000), grl=feat, singleEnd=TRUE, readle
p <- plotfeaturetypeCounts(x=fc, graphicsfile="results/featureCounts.pdf", graphicsformat="pdf", scales="f
```

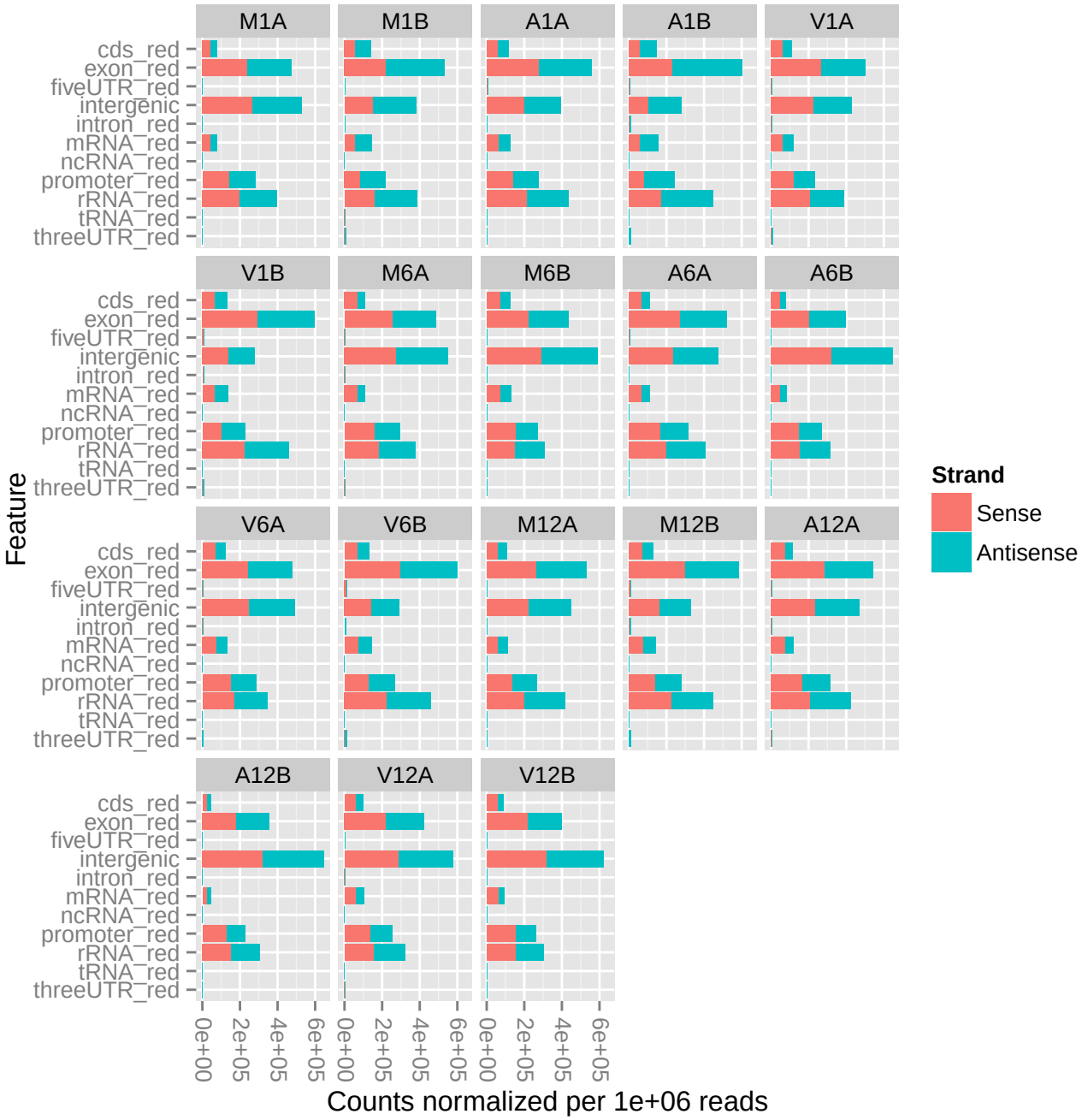

Figure 2: Read distribution plot across annotation features for any read length.

5.3 Count and plot reads of specific lengths

Generate and plot feature counts for specific read lengths

```
fc2 <- featuretypeCounts(bfl=BamFileList(outpaths(args), yieldSize=50000), grl=feat, singleEnd=TRUE, readl
p2 <- plotfeaturetypeCounts(x=fc2, graphicsfile="results/featureCounts2.pdf", graphicsformat="pdf", scales
```

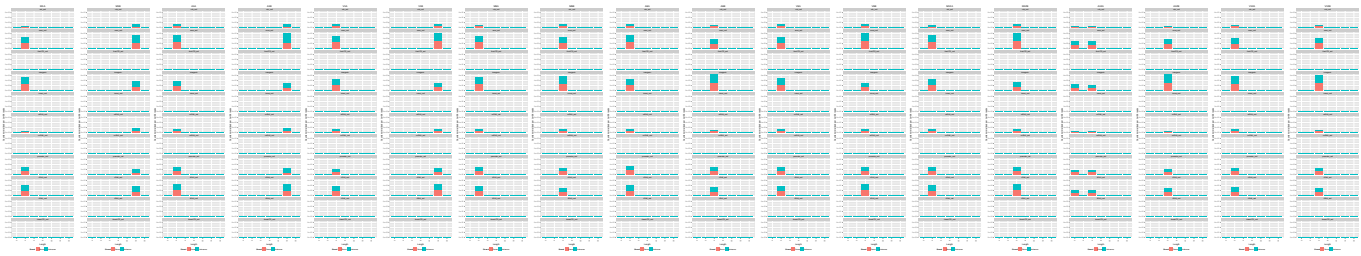

Figure 3: Read distribution plot across annotation features for specific read lengths.

6 Adding custom features to workflow

6.1 Predicting uORFs in 5' UTR regions

The function `predORF` can be used to identify open reading frames (ORFs) and coding sequences (CDSs) in DNA sequences provided as `DNAString` or `DNAStringSet` objects. The setting `mode='ORF'` returns continuous reading frames that begin with a start codon and end with a stop codon, while `mode='CDS'` returns continuous reading frames that do not need to begin or end with start or stop codons, respectively. The argument `n` defines the maximum number of ORFs to return for each input sequence (*e.g.* `n=1` returns only the longest ORF). If `n="all"` and `longest_disjoint=TRUE`, then the function will return all non-overlapping ORFs including the longest ORF.

```
library(systemPipeRdata); library(GenomicFeatures); library(rtracklayer)
gff <- system.file("extdata/annotation", "tair10.gff", package="systemPipeRdata")
txdb <- makeTxDbFromGFF(file=gff, format="gff3", organism="Arabidopsis")
futr <- fiveUTRsByTranscript(txdb, use.names=TRUE)
genome <- system.file("extdata/annotation", "tair10.fasta", package="systemPipeRdata")
dna <- extractTranscriptSeqs(FaFile(genome), futr)
uorf <- predORF(dna, n="all", mode="orf", longest_disjoint=TRUE, strand="sense")
```

To use the predicted ORF ranges for expression analysis given genome alignments as input, it is necessary to scale them to the corresponding genome coordinates. The function `scaleRanges` does this by transforming the mappings of spliced features (query ranges) to their corresponding genome coordinates (subject ranges). The method accounts for introns in the subject ranges that are absent in the query ranges. The above uORFs predicted in the provided 5' UTRs sequences using `predORF` are a typical use case for this application. These query ranges are given relative to the 5' UTR sequences and `scaleRanges` will convert them to the corresponding genome coordinates.

```
grl_scaled <- scaleRanges(subject=futr, query=uorf, type="uORF", verbose=TRUE)
export.gff3(unlist(grl_scaled), "uorf.gff")
```

7 Genomic read coverage along transripts or CDSs

The `featureCoverage` function computes the read coverage along single and multi component features based on genomic alignments. The coverage segments of component features are spliced to continuous ranges, such as exons to transcripts or CDSs to ORFs. The results can be obtained with single nucleotide resolution (*e.g.* around start and stop codons) or as mean coverage of relative bin sizes, such as 100 bins for each feature. The latter allows comparisons of coverage trends among transcripts of variable length. Additionally, the results can be obtained for single or many

features (*e.g.* any number of transcritpts) at once. Visualization of the coverage results is facilitated by the downstream `plotfeatureCoverage` function.

7.1 Binned CDS coverage to compare many transcripts

```
grl <- cdsBy(txdb, "tx", use.names=TRUE)
fcov <- featureCoverage(bfl=BamFileList(outpaths(args)[1:2]), grl=grl[1:4], resizereads=NULL,
                        readlengthrange=NULL, Nbins=20, method=mean, fixedmatrix=FALSE,
                        resizefeatures=TRUE, upstream=20, downstream=20,
                        outfile="results/featureCoverage.xls", overwrite=TRUE)
```

7.2 Coverage upstream and downstream of start and stop codons

```
fcov <- featureCoverage(bfl=BamFileList(outpaths(args)[1:4]), grl=grl[1:12], resizereads=NULL,
                        readlengthrange=NULL, Nbins=NULL, method=mean, fixedmatrix=TRUE,
                        resizefeatures=TRUE, upstream=20, downstream=20,
                        outfile="results/featureCoverage.xls", overwrite=TRUE)
plotfeatureCoverage(covMA=fcov, method=mean, scales="fixed", extendylim=2, scale_count_val=10^6)
```

7.3 Combined coverage for both binned CDS and start/stop codons

```
fcov <- featureCoverage(bfl=BamFileList(outpaths(args)[1:2]), grl=grl[1:4], resizereads=NULL,
                        readlengthrange=NULL, Nbins=20, method=mean, fixedmatrix=TRUE,
                        resizefeatures=TRUE, upstream=20, downstream=20,
                        outfile="results/featureCoverage.xls", overwrite=TRUE)
pdf("./results/featurePlot.pdf", height=12, width=24)
plotfeatureCoverage(covMA=fcov, method=mean, scales="fixed", extendylim=2, scale_count_val=10^6)
dev.off()
```

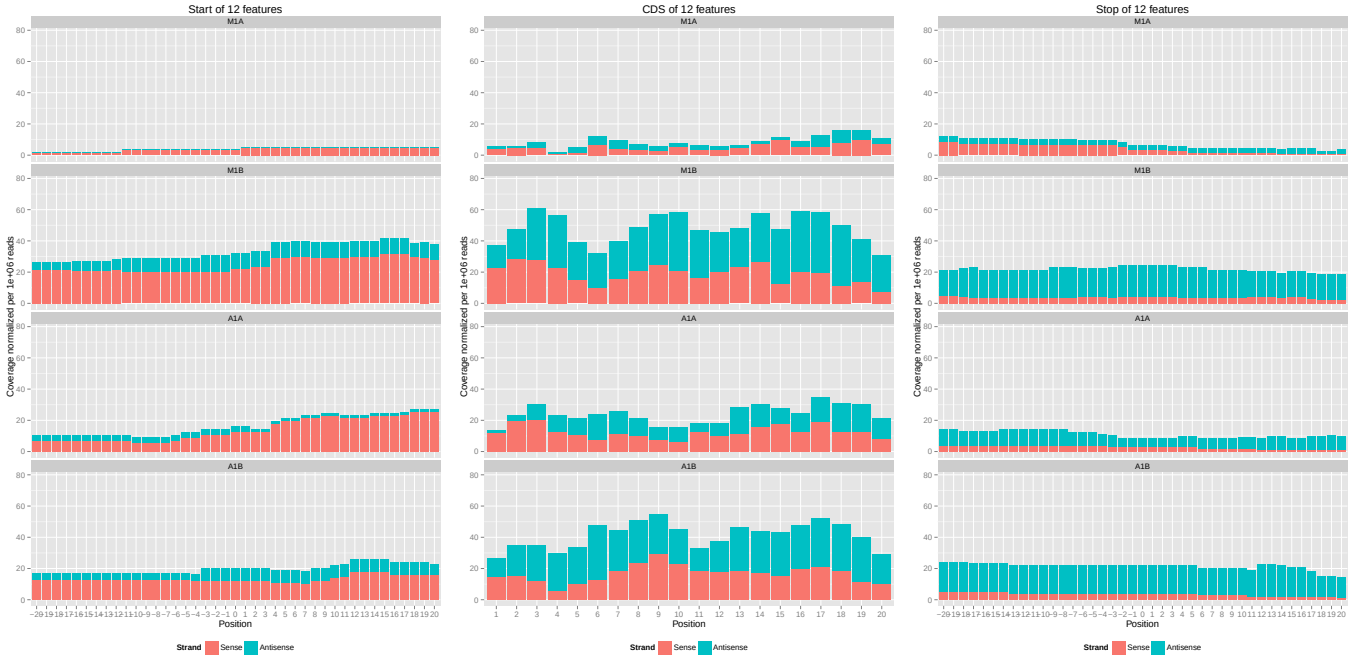

Figure 4: Feature coverage plot with single nucleotide resolution around start and stop codons and binned coverage between them.

7.4 Nucleotide level coverage along entire transcripts/CDSs

```
fcov <- featureCoverage(bfl=BamFileList(outpaths(args)[1:2]), grl=grl[1:4], resizereads=NULL,
                        readlengthrange=NULL, Nbins=NULL, method=mean, fixedmatrix=FALSE,
                        resizefeatures=TRUE, upstream=20, downstream=20)
plotfeatureCoverage(covMA=fcov, method=mean, scales="fixed", scale_count_val=10^6)
```

8 Read quantification per annotation range

8.1 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 correspoding RPKM table (rpkmDFeByg.xls) are written to separate files in the `results` directory of this project. Parallelization is achieved with the *BiocParallel* package, here using 8 CPU cores.

```
library("GenomicFeatures"); library(BiocParallel)
txdb <- loadDb("./data/tair10.sqlite")
eByg <- exonsBy(txdb, by=c("gene"))
bfl <- BamFileList(outpaths(args), yieldSize=50000, index=character())
multicoreParam <- MulticoreParam(workers=8); 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.

8.2 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 (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=targetsin(args)$SampleName, condition=targetsin(args)$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()
```

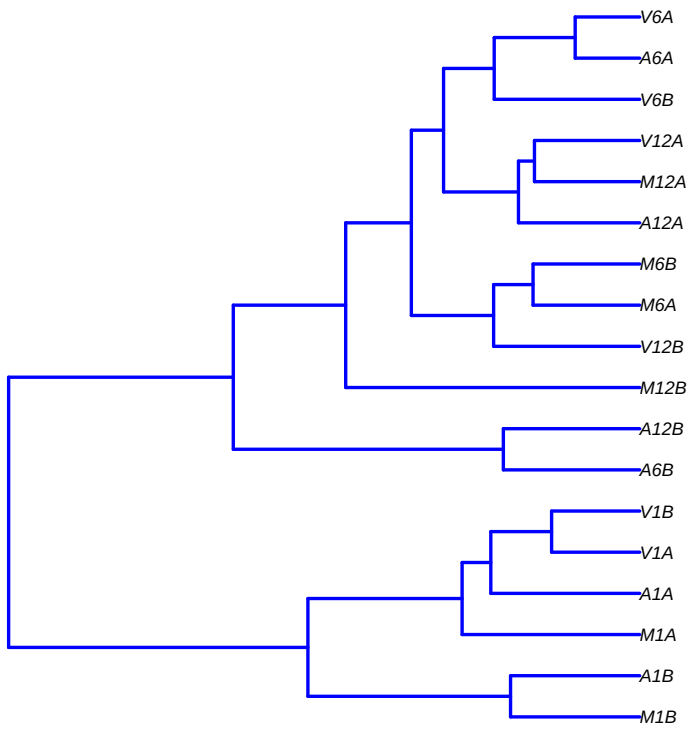

Figure 5: Correlation dendrogram of samples.

9 Analysis of differentially expressed genes with edgeR

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

```
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 custom functional descriptions. Skip this step if `desc.xls` is not available.

```
desc <- read.delim("data/desc.xls")
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)
```

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=1))
dev.off()
```

```
write.table(DEG_list$Summary, "./results/DEGcounts.xls", quote=FALSE, sep="\t", row.names=FALSE)
```

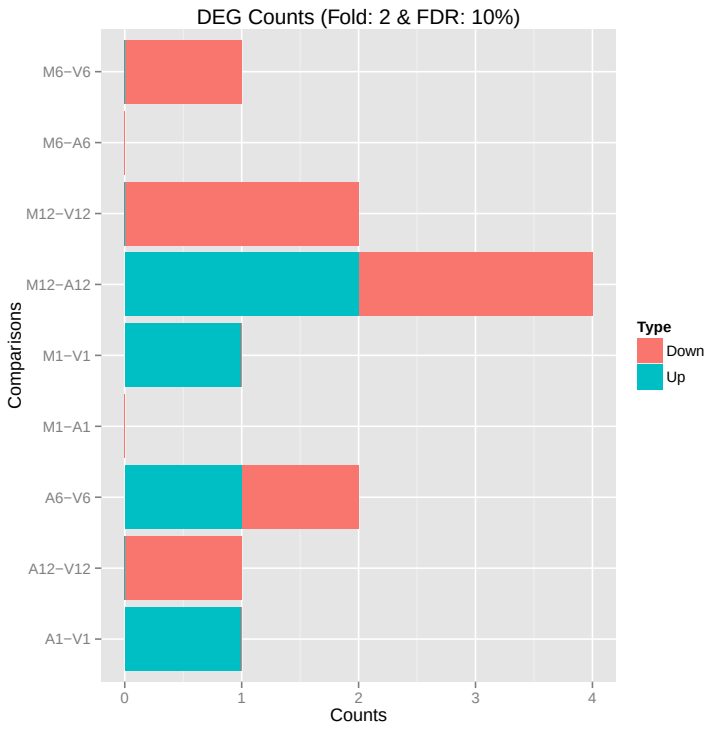

Figure 6: Up and down regulated DEGs with FDR of 1%.

The function `overLapper` can compute Venn intersects for large numbers of sample sets (up to 20 or more) and `vennPlot` can plot 2-5 way Venn diagrams. A useful feature is the possiblity 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()
```

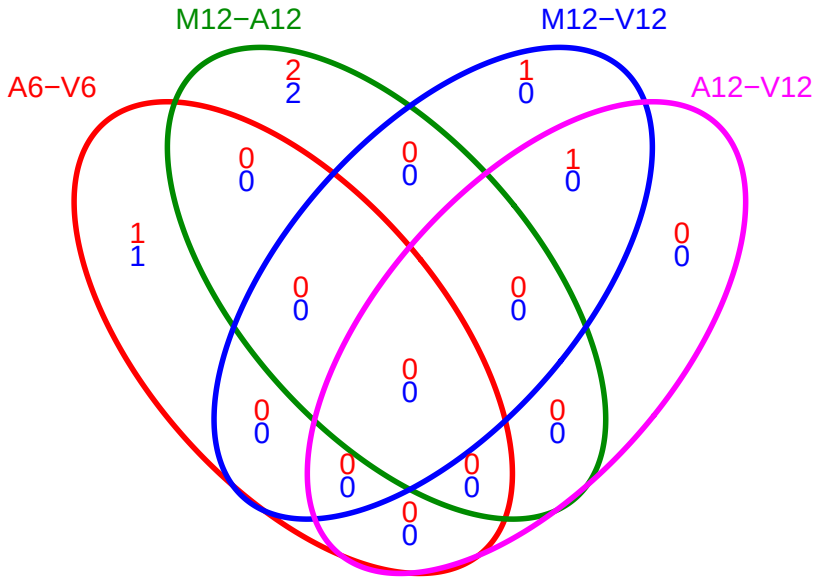

Figure 7: Venn Diagram for 4 Up and Down DEG Sets.

9.1 GO term enrichment analysis of DEGs

9.1.1 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
m <- useMart("ENSEMBL_MART_PLANT"); listDatasets(m)
m <- useMart("ENSEMBL_MART_PLANT", dataset="athaliana_eg_gene")
listAttributes(m) # Choose data types you want to download
go <- getBM(attributes=c("go_accession", "tair_locus", "go_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]=="cellul
go[1:4,]
dir.create("./data/GO")
write.table(go, "data/GO/GOannotationsBiomart_mod.txt", quote=FALSE, row.names=FALSE, col.names=FALSE, sep
catdb <- makeCATdb(myfile="data/GO/GOannotationsBiomart_mod.txt", lib=NULL, org="", colno=c(1,2,3), idconv
save(catdb, file="data/GO/catdb.RData")
```

9.1.2 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 `GOCluster_Report` 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.

```
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
library("biomaRt"); m <- useMart("ENSEMBL_MART_PLANT", dataset="athaliana_eg_gene")
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=g
```

9.1.3 Plot batch GO term results

The `data.frame` generated by `GOCluster_Report` 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")
```

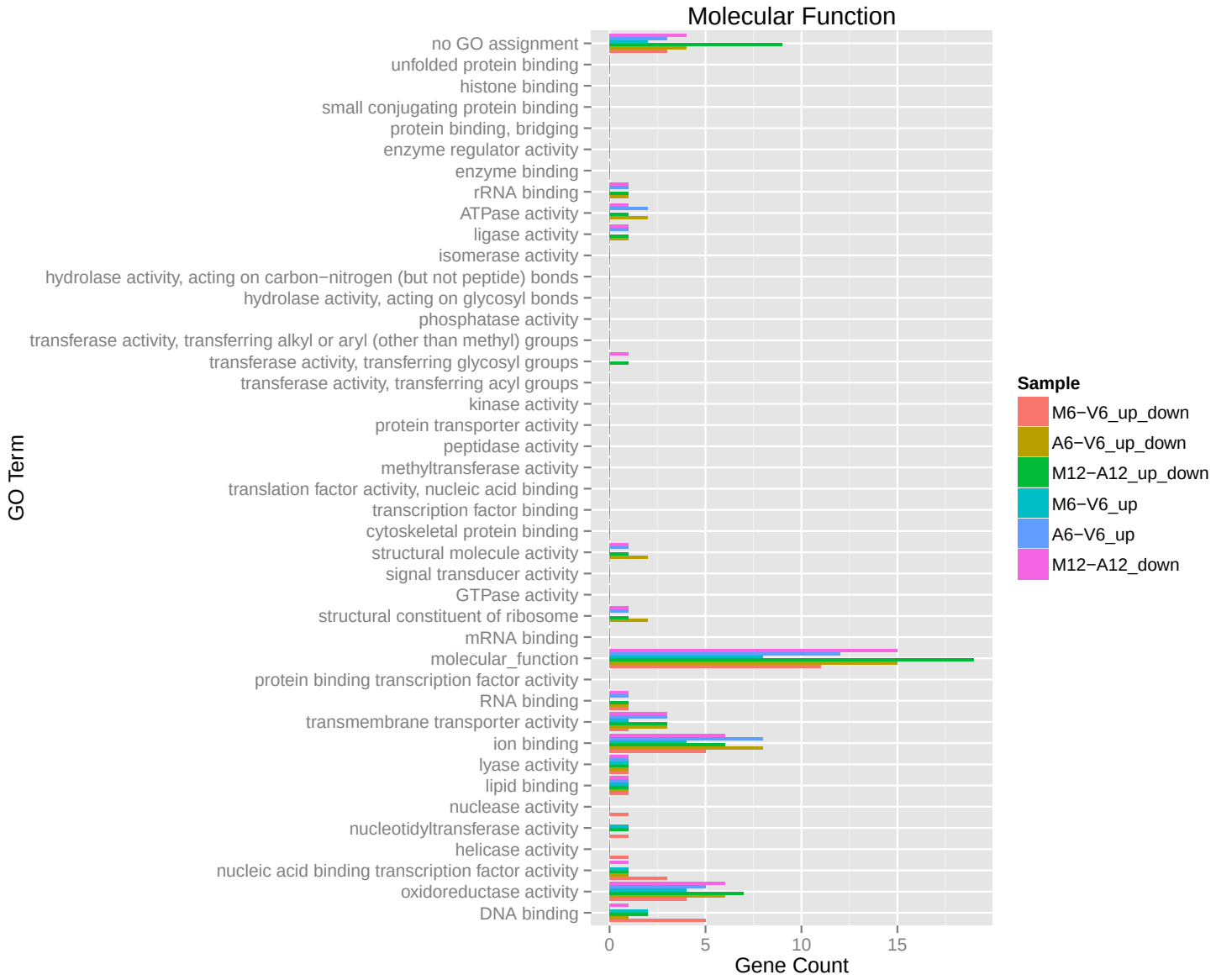

Figure 8: GO Slim Barplot for MF Ontology.

10 Differential ribosome binding analysis (tranlational efficiency)

To be added...

11 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()
```

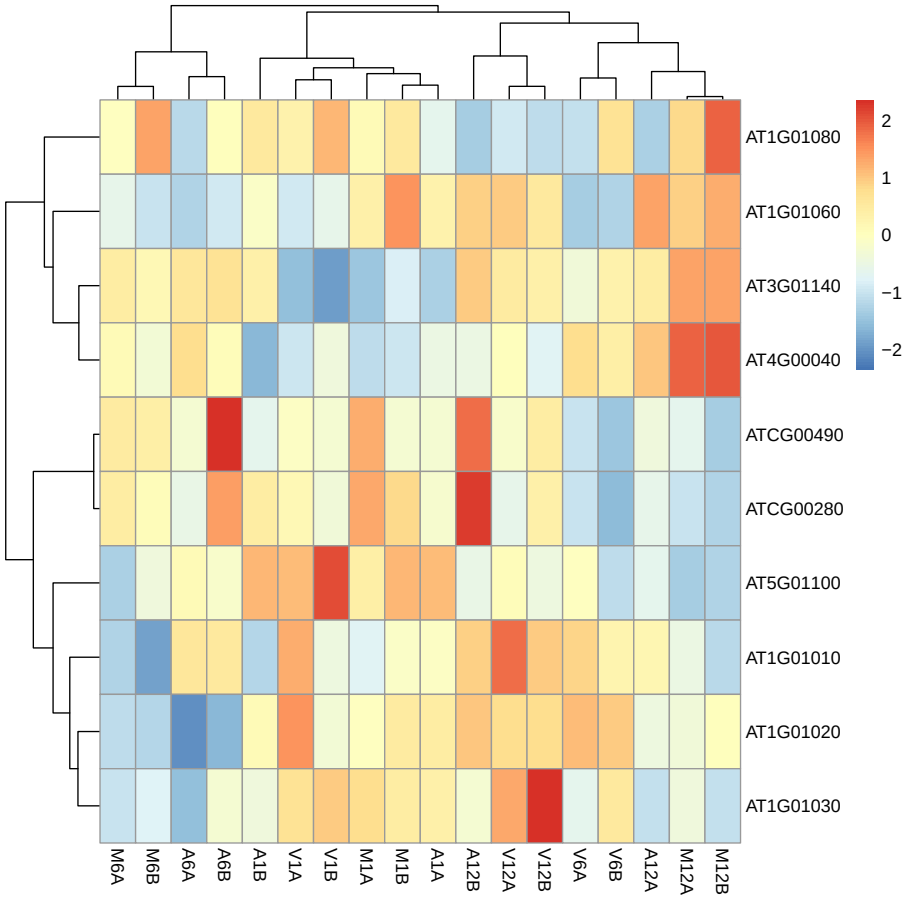

Figure 9: Heat map with hierarchical clustering dendrograms of DEGs.

12 Version Information

```
toLatex(sessionInfo())
```

- R version 3.2.2 (2015-08-14), x86_64-pc-linux-gnu
- Locale: LC_CTYPE=en_US.UTF-8, LC_NUMERIC=C, LC_TIME=en_US.UTF-8, LC_COLLATE=en_US.UTF-8, LC_MONETARY=en_US.UTF-8, LC_MESSAGES=en_US.UTF-8, LC_PAPER=en_US.UTF-8, LC_NAME=C, LC_ADDRESS=C, LC_TELEPHONE=C, LC_MEASUREMENT=en_US.UTF-8, LC_IDENTIFICATION=C
- Base packages: base, datasets, graphics, grDevices, methods, parallel, stats, stats4, utils
- Other packages: Biobase 2.29.1, BiocGenerics 0.15.9, BiocParallel 1.3.52, Biostrings 2.37.8, DBI 0.3.1, GenomeInfoDb 1.5.16, GenomicAlignments 1.5.18, GenomicRanges 1.21.29, IRanges 2.3.23, knitr 1.11, Rsamtools 1.21.19, RSQLite 1.0.0, S4Vectors 0.7.22, ShortRead 1.27.5, SummarizedExperiment 0.3.9, systemPipeR 1.3.46, XVector 0.9.4

- Loaded via a namespace (and not attached): annotate 1.47.4, AnnotationDbi 1.31.18, AnnotationForge 1.11.20, base64enc 0.1-3, BatchJobs 1.6, BBmisc 1.9, BiocStyle 1.7.7, biomaRt 2.25.3, bitops 1.0-6, brew 1.0-6, Category 2.35.1, checkmate 1.6.2, colorspace 1.2-6, digest 0.6.8, edgeR 3.11.4, evaluate 0.8, fail 1.3, formatR 1.2.1, futile.logger 1.4.1, futile.options 1.0.0, genefilter 1.51.1, GenomicFeatures 1.21.31, ggplot2 1.0.1, GO.db 3.2.1, GOstats 2.35.1, graph 1.47.2, grid 3.2.2, GSEABase 1.31.3, gtable 0.1.2, highr 0.5.1, hwriter 1.3.2, lambda.r 1.1.7, lattice 0.20-33, latticeExtra 0.6-26, limma 3.25.16, magrittr 1.5, MASS 7.3-43, Matrix 1.2-2, munsell 0.4.2, pheatmap 1.0.7, plyr 1.8.3, proto 0.3-10, RBGL 1.45.1, RColorBrewer 1.1-2, Rcpp 0.12.1, RCurl 1.95-4.7, reshape2 1.4.1, rjson 0.2.15, rtracklayer 1.29.28, scales 0.3.0, sendmailR 1.2-1, splines 3.2.2, stringi 0.5-5, stringr 1.0.0, survival 2.38-3, tools 3.2.2, XML 3.98-1.3, xtable 1.7-4, zlibbioc 1.15.0

13 Funding

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

14 References

Thomas Girke. systemPipeR: NGS workflow and report generation environment, 28 June 2014. URL https://github.com/tgirke/systemPipeR.

Piyada Juntawong, Thomas Girke, Jérémie Bazin, and Julia Bailey-Serres. Translational dynamics revealed by genome-wide profiling of ribosome footprints in arabidopsis. *Proc. Natl. Acad. Sci. U. S. A.*, 111(1):E203–12, 7 January 2014. ISSN 0027-8424, 1091-6490. doi: 10.1073/pnas.1317811111. URL http://dx.doi.org/10.1073/pnas.1317811111.

Daehwan Kim, Geo Pertea, Cole Trapnell, Harold Pimentel, Ryan Kelley, and Steven L Salzberg. TopHat2: accurate alignment of transcriptomes in the presence of insertions, deletions and gene fusions. *Genome Biol.*, 14(4):R36, 25 April 2013. ISSN 1465-6906. doi: 10.1186/gb-2013-14-4-r36. URL http://dx.doi.org/10.1186/gb-2013-14-4-r36.

Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with bowtie 2. *Nat. Methods*, 9(4):357–359, April 2012. ISSN 1548-7091. doi: 10.1038/nmeth.1923. URL http://dx.doi.org/10.1038/nmeth.1923.

Michael Lawrence, Wolfgang Huber, Hervé Pagès, Patrick Aboyoun, Marc Carlson, Robert Gentleman, Martin T Morgan, and Vincent J Carey. Software for computing and annotating genomic ranges. *PLoS Comput. Biol.*, 9(8):e1003118, 8 August 2013. ISSN 1553-734X. doi: 10.1371/journal.pcbi.1003118. URL http://dx.doi.org/10.1371/journal.pcbi.1003118.

M D Robinson, D J McCarthy, and G K Smyth. edger: a bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics*, 26(1):139–140, January 2010. ISSN 1367-4803. doi: 10.1093/bioinformatics/btp616. URL http://dx.doi.org/10.1093/bioinformatics/btp616.