systemPipeR: NGS workflow and report generation environment

Thomas Girke
Email contact: thomas.girke@ucr.edu

July 15, 2015

1 Introduction

systemPipeR provides utilities for building *end-to-end* analysis workflows with automated report generation for next generation sequence (NGS) applications such as RNA-Seq, ChIP-Seq, VAR-Seq and many others (Girke, 2014). An important feature is support for running command-line software, such as NGS aligners, on both single machines or compute clusters. This includes both interactive job submissions or batch submissions to queuing systems of clusters. For instance, *systemPipeR* can be used with most command-line aligners such as `BWA` (Li, 2013; Li and Durbin, 2009), `TopHat 2` (Kim et al., 2013) and `Bowtie 2` (Langmead and Salzberg, 2012), as well as the R-based NGS aligners *Rsubread* (Liao et al., 2013) and *gsnap* (Wu and Nacu, 2010). Efficient handling of complex sample sets and experimental designs is facilitated by a well-defined sample annotation infrastructure which improves reproducibility and user-friendliness of many typical analysis workflows in the NGS area (Lawrence et al., 2013).

A central concept for designing workflows within the *sytemPipeR* environment is the use of sample management containers called `SYSargs`. Instances of this S4 object class are constructed by the `systemArgs` function from two simple tablular files: a `targets` file and a `param` file. The latter is optional for workflow steps lacking command-line software. Typically, a `SYSargs` instance stores all sample-level inputs as well as the paths to the corresponding outputs generated by command-line- or R-based software generating sample-level output files, such as read preprocessors (trimmed/filtered FASTQ files), aligners (SAM/BAM files), variant callers (VCF/BCF files) or peak callers (BED/WIG files). Each sample level input/outfile operation uses its own `SYSargs` instance. The outpaths of `SYSargs` usually define the sample inputs for the next `SYSargs` instance. This connectivity is established by writing the outpaths with the `writeTargetsout` function to a new targets file that serves as input to the next `systemArgs` call. By chaining several `SYSargs` steps together one can construct complex workflows involving many sample-level input/output file operations with any combinaton of command-line or R-based software.

The intended way of running *sytemPipeR* workflows is via `*.Rnw` or `*.Rmd` files, which can be executed either line-wise in interactive mode or with a single command from R or the command-line using a Makefile. This way comprehensive and reproducible analysis reports in PDF or HTML format can be generated in a fully automated manner. Templates for setting up custom project reports are provided as `*.Rnw` files in the `vignettes` subdirectory of this package. The corresponding PDFs of these report templates are linked here: systemPipeRNAseq, systemPipeChIPseq and systemPipeVARseq. To work with `*.Rnw` or `*.Rmd` files efficiently, basic knowledge of Sweave or knitr and Latex or Markdown is required.

Contents

2 Getting Started

2.1 Installation

The R software for running *systemPipeR* can be downloaded from CRAN (http://cran.at.r-project.org/). The *systemPipeR* package can be installed from R using the `biocLite` install command.

```
source("http://bioconductor.org/biocLite.R") # Sources the biocLite.R installation script
biocLite("systemPipeR") # Installs the package
```

2.2 Loading the Package and Documentation

```
library("systemPipeR") # Loads the package
library(help="systemPipeR") # Lists all functions and classes
vignette("systemPipeR") # Opens this PDF manual from R
```

2.3 Sample FASTQ Files

The mini sample FASTQ files used by this overview vignette as well as the associated workflow reporting vignettes 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 annotion 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.

3 Structure of `targets` file

The `targets` file defines all input files (*e.g.* FASTQ, BAM, BCF) and sample comparisons of an analysis workflow. The following shows the format of a sample `targets` file provided by this package. In a target file with a single type of input files, here FASTQ files of single end (SE) reads, the first three columns are mandatory including their column names, while it is four mandatory columns for FASTQ files for PE reads. All subsequent columns are optional and any number of additional columns can be added as needed.

```
library(systemPipeR)
targetspath <- system.file("extdata", "targets.txt", package="systemPipeR")
read.delim(targetspath, comment.char = "#")

                   FileName SampleName Factor SampleLong Experiment        Date
1  ./data/SRR446027_1.fastq        M1A     M1  Mock.1h.A          1 23-Mar-2012
2  ./data/SRR446028_1.fastq        M1B     M1  Mock.1h.B          1 23-Mar-2012
3  ./data/SRR446029_1.fastq        A1A     A1   Avr.1h.A          1 23-Mar-2012
4  ./data/SRR446030_1.fastq        A1B     A1   Avr.1h.B          1 23-Mar-2012
5  ./data/SRR446031_1.fastq        V1A     V1   Vir.1h.A          1 23-Mar-2012
6  ./data/SRR446032_1.fastq        V1B     V1   Vir.1h.B          1 23-Mar-2012
7  ./data/SRR446033_1.fastq        M6A     M6  Mock.6h.A          1 23-Mar-2012
8  ./data/SRR446034_1.fastq        M6B     M6  Mock.6h.B          1 23-Mar-2012
9  ./data/SRR446035_1.fastq        A6A     A6   Avr.6h.A          1 23-Mar-2012
10 ./data/SRR446036_1.fastq        A6B     A6   Avr.6h.B          1 23-Mar-2012
11 ./data/SRR446037_1.fastq        V6A     V6   Vir.6h.A          1 23-Mar-2012
12 ./data/SRR446038_1.fastq        V6B     V6   Vir.6h.B          1 23-Mar-2012
13 ./data/SRR446039_1.fastq       M12A    M12 Mock.12h.A          1 23-Mar-2012
14 ./data/SRR446040_1.fastq       M12B    M12 Mock.12h.B          1 23-Mar-2012
15 ./data/SRR446041_1.fastq       A12A    A12  Avr.12h.A          1 23-Mar-2012
16 ./data/SRR446042_1.fastq       A12B    A12  Avr.12h.B          1 23-Mar-2012
17 ./data/SRR446043_1.fastq       V12A    V12  Vir.12h.A          1 23-Mar-2012
18 ./data/SRR446044_1.fastq       V12B    V12  Vir.12h.B          1 23-Mar-2012
```

Structure of `targets` file for paired end (PE) samples.

```
targetspath <- system.file("extdata", "targetsPE.txt", package="systemPipeR")
read.delim(targetspath, comment.char = "#")[1:2,1:6]

                 FileName1                FileName2 SampleName Factor SampleLong Experiment
1 ./data/SRR446027_1.fastq ./data/SRR446027_2.fastq        M1A     M1  Mock.1h.A          1
2 ./data/SRR446028_1.fastq ./data/SRR446028_2.fastq        M1B     M1  Mock.1h.B          1
```

Sample comparisons are defined in the header lines of the `targets` file starting with '`# <CMP>`'. The function `readComp` imports the comparison and stores them in a `list`. Alternatively, `readComp` can obtain the comparison information from the corresponding `SYSargs` object (see below). Note, the header lines are optional in `targets` files. They are mainly useful for controlling comparative analysis according to certain biological expectations, such as simple pairwise comparisons in RNA-Seq experiments.

```
readComp(file=targetspath, format="vector", delim="-")

$CMPset1
[1] "M1-A1"   "M1-V1"   "A1-V1"   "M6-A6"   "M6-V6"   "A6-V6"   "M12-A12" "M12-V12" "A12-V12"

$CMPset2
 [1] "M1-A1"   "M1-V1"   "M1-M6"   "M1-A6"   "M1-V6"   "M1-M12"  "M1-A12"  "M1-V12"  "A1-V1"
[10] "A1-M6"   "A1-A6"   "A1-V6"   "A1-M12"  "A1-A12"  "A1-V12"  "V1-M6"   "V1-A6"   "V1-V6"
[19] "V1-M12"  "V1-A12"  "V1-V12"  "M6-A6"   "M6-V6"   "M6-M12"  "M6-A12"  "M6-V12"  "A6-V6"
[28] "A6-M12"  "A6-A12"  "A6-V12"  "V6-M12"  "V6-A12"  "V6-V12"  "M12-A12" "M12-V12" "A12-V12"
```

4 Structure of param file and SYSargs container

The param file defines the parameters of the command-line software. The following shows the format of a sample param file provided by this package.

```
parampath <- system.file("extdata", "tophat.param", package="systemPipeR")
read.delim(parampath, comment.char = "#")

     PairSet        Name                              Value
1    modules        <NA>                        bowtie2/2.1.0
2    modules        <NA>                        tophat/2.0.8b
3   software        <NA>                               tophat
4      cores          -p                                    4
5      other        <NA> -g 1 --segment-length 25 -i 30 -I 3000
6   outfile1          -o                          <FileName1>
7   outfile1        path                           ./results/
8   outfile1      remove                                 <NA>
9   outfile1      append                              .tophat
10  outfile1 outextension            .tophat/accepted_hits.bam
11 reference        <NA>                  ./data/tair10.fasta
12   infile1        <NA>                          <FileName1>
13   infile1        path                                 <NA>
14   infile2        <NA>                          <FileName2>
15   infile2        path                                 <NA>
```

The systemArgs function imports the definitions of both the param file and the targets file, and stores all relevant information as SYSargs object. To run the pipeline without command-line software, one can assign NULL to sysma instead of a param file. In addition, one can start the *systemPipeR* workflow with pregenerated BAM files by providing a targets file where the FileName column gives the paths to the BAM files and sysma is assigned NULL.

```
args <- systemArgs(sysma=parampath, mytargets=targetspath)
args

An instance of 'SYSargs' for running 'tophat' on 18 samples
```

Several accessor functions are available that are named after the slot names of the SYSargs object class.

```
names(args)

 [1] "targetsin"     "targetsout"    "targetsheader" "modules"       "software"
 [6] "cores"         "other"         "reference"     "results"       "infile1"
[11] "infile2"       "outfile1"      "sysargs"       "outpaths"
```

```
modules(args)

[1] "bowtie2/2.1.0" "tophat/2.0.8b"

cores(args)

[1] 4

outpaths(args)[1]

"/rhome/tgirke/Projects/github/systemPipeR_dev/systemPipeR/vignettes/results/SRR446027_1.fastq.tophat/acce

sysargs(args)[1]

"tophat -p 4 -g 1 --segment-length 25 -i 30 -I 3000 -o /rhome/tgirke/Projects/github/systemPipeR_dev/syste
```

The content of the param file can be returned as JSON object as follows (requires *rjson* package).

```
systemArgs(sysma=parampath, mytargets=targetspath, type="json")

[1] "{\"modules\":{\"n1\":\"\",\"v2\":\"bowtie2/2.1.0\",\"n1\":\"\",\"v2\":\"tophat/2.0.8b\"},\"software\"
```

5 Workflow

5.1 Define environment settings and samples

Load package:

```
library(systemPipeR)
```

Construct SYSargs object from param and targets files.

```
args <- systemArgs(sysma="trim.param", mytargets="targets.txt")
```

5.2 Read Preprocessing

The function preprocessReads allows to apply predefined or custom read preprocessing functions to all FASTQ files referenced in a SYSargs container, such as quality filtering or adaptor trimming routines. The paths to the resulting output FASTQ files are stored in the outpaths slot of the SYSargs object. Internally, preprocessReads uses the FastqStreamer function from the *ShortRead* package to stream through large FASTQ files in a memory-efficient manner. The following example performs adaptor 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 SYSargs instance, *e.g.* running the NGS alignments using the trimmed FASTQ files.

```
preprocessReads(args=args, Fct="trimLRPatterns(Rpattern='GCCCGGGTAA', subject=fq)",
                batchsize=100000, overwrite=TRUE, compress=TRUE)
writeTargetsout(x=args, file="targets_trim.txt")
```

The following example shows how one can design a custom read preprocessing function using utilities provided by the *ShortRead* package, and then run it in batch mode with the preprocessReads function (here on paired-end reads).

```
args <- systemArgs(sysma="trimPE.param", mytargets="targetsPE.txt")
filterFct <- function(fq, cutoff=20, Nexceptions=0) {
    qcount <- rowSums(as(quality(fq), "matrix") <= cutoff)
```

```
    fq[qcount <= Nexceptions] # Retains reads where Phred scores are >= cutoff with N exceptions
}
preprocessReads(args=args, Fct="filterFct(fq, cutoff=20, Nexceptions=0)", batchsize=100000)
writeTargetsout(x=args, file="targets_PEtrim.txt")
```

5.3 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.

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

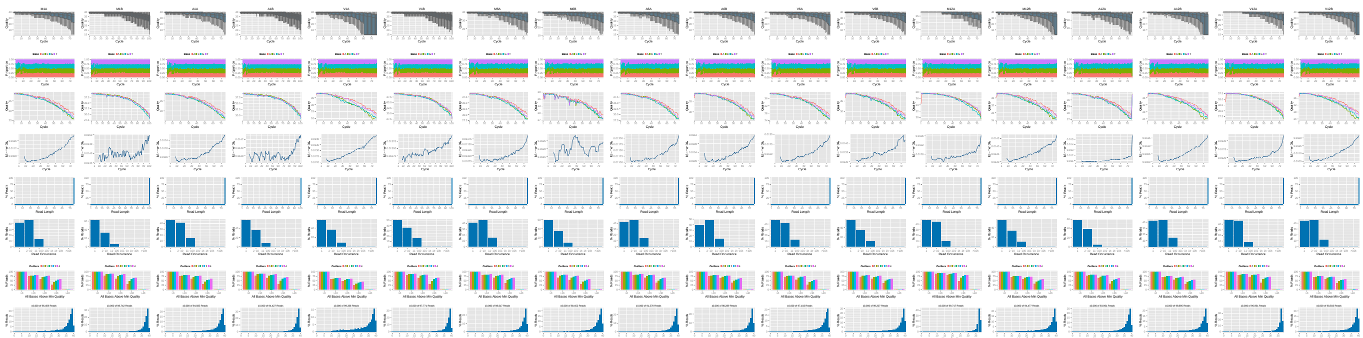

Figure 1: QC report for 18 FASTQ files.

Parallelization of QC report on single machine with multiple cores

```
args <- systemArgs(sysma="tophat.param", mytargets="targets.txt")
f <- function(x) seeFastq(fastq=infile1(args)[x], batchsize=100000, klength=8)
fqlist <- bplapply(seq(along=args), f, BPPARAM = MulticoreParam(workers=8))
seeFastqPlot(unlist(fqlist, recursive=FALSE))
```

Parallelization of QC report via scheduler (*e.g.* Torque) across several compute nodes

```
library(BiocParallel); library(BatchJobs)
f <- function(x) {
    library(systemPipeR)
    args <- systemArgs(sysma="tophat.param", mytargets="targets.txt")
    seeFastq(fastq=infile1(args)[x], batchsize=100000, klength=8)
}
funs <- makeClusterFunctionsTorque("torque.tmpl")
param <- BatchJobsParam(length(args), resources=list(walltime="20:00:00", nodes="1:ppn=1", memory="6gb"),
register(param)
fqlist <- bplapply(seq(along=args), f)
seeFastqPlot(unlist(fqlist, recursive=FALSE))
```

5.4 Alignment with Tophat 2

Build Bowtie 2 index.

```
args <- systemArgs(sysma="tophat.param", mytargets="targets.txt")
moduleload(modules(args)) # Skip if module system is not available
system("bowtie2-build ./data/tair10.fasta ./data/tair10.fasta")
```

Execute SYSargs on a single machine without submitting to a queuing system of a compute cluster. This way the input FASTQ files will be processed sequentially. If available, multiple CPU cores can be used for processing each file. The number of CPU cores (here 4) to use for each process is defined in the *.param file. With cores(args) one can return this value from the SYSargs object. Note, if a module system is not installed or used, then the corresponding *.param file needs to be edited accordingly by either providing an empty field in the line(s) starting with module or by deleting these lines.

```
bampaths <- runCommandline(args=args)
```

Alternatively, the computation can be greatly accelerated by processing many files in parallel using several compute nodes of a cluster, where a scheduling/queuing system is used for load balancing. To avoid over-subscription of CPU cores on the compute nodes, the value from cores(args) is passed on to the submission command, here nodes in the resources list object. The number of independent parallel cluster processes is defined under the Njobs argument. The following example will run 18 processes in parallel using for each 4 CPU cores. If the resources available on a cluster allow to run all 18 processes at the same time then the shown sample submission will utilize in total 72 CPU cores. Note, runCluster can be used with most queueing systems as it is based on utilities from the BatchJobs package which supports the use of template files (*.tmpl) for defining the run parameters of different schedulers. To run the following code, one needs to have both a conf file (see .BatchJob samples here) and a template file (see *.tmpl samples here) for the queueing available on a system. The following example uses the sample conf and template files for the Torque scheduler provided by this package.

```
file.copy(system.file("extdata", ".BatchJobs.R", package="systemPipeR"), ".")
file.copy(system.file("extdata", "torque.tmpl", package="systemPipeR"), ".")
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)
```

Useful commands for monitoring progress of submitted jobs

```
showStatus(reg)
file.exists(outpaths(args))
sapply(1:length(args), function(x) loadResult(reg, x)) # Works after job completion
```

5.5 Read and alignment count stats

Generate table of read and alignment counts for all samples.

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

The following shows the first four lines of the sample alignment stats file provided by the *systemPipeR* package. For simplicity the number of PE reads is multiplied here by 2 to approximate proper alignment frequencies where each read in a pair is counted.

```
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
```

Parallelization of read/alignment stats on single machine with multiple cores

```
f <- function(x) alignStats(args[x])
read_statsList <- bplapply(seq(along=args), f, BPPARAM = MulticoreParam(workers=8))
read_statsDF <- do.call("rbind", read_statsList)
```

Parallelization of read/alignment stats via scheduler (e.g. Torque) across several compute nodes

```
library(BiocParallel); library(BatchJobs)
f <- function(x) {
    library(systemPipeR)
    args <- systemArgs(sysma="tophat.param", mytargets="targets.txt")
    alignStats(args[x])
}
funs <- makeClusterFunctionsTorque("torque.tmpl")
param <- BatchJobsParam(length(args), resources=list(walltime="20:00:00", nodes="1:ppn=1", memory="6gb"),
register(param)
read_statsList <- bplapply(seq(along=args), f)
read_statsDF <- do.call("rbind", read_statsList)
```

5.6 Create symbolic links for viewing BAM files in IGV

The genome browser IGV supports reading of indexed/sorted BAM files via web URLs. This way it can be avoided to create unnecessary copies of these large files. To enable this approach, an HTML directory with http access needs to be available in the user account (*e.g.* `~/public_html`) of a system. If this is not the case then the BAM files need to be moved or copied to the system where IGV runs. In the following, `htmldir` defines the path to the HTML directory with http access where the symbolic links to the BAM files will be stored. The corresponding URLs will be written to a text file specified under the `urlfile` argument.

```
symLink2bam(sysargs=args, htmldir=c("~/.html/", "somedir/"),
            urlbase="http://myserver.edu/~username/",
            urlfile="IGVurl.txt")
```

5.7 Alternative NGS Aligners

5.7.1 Alignment with Bowtie 2 (e.g. for miRNA profiling)

The following example runs Bowtie 2 as a single process without submitting it to a cluster.

```
args <- systemArgs(sysma="bowtieSE.param", mytargets="targets.txt")
moduleload(modules(args)) # Skip if module system is not available
bampaths <- runCommandline(args=args)
```

Alternatively, submit the job to compute nodes.

```
qsubargs <- getQsubargs(queue="batch", cores=cores(args), memory="mem=10gb", time="walltime=20:00:00")
(joblist <- qsubRun(args=args, qsubargs=qsubargs, Nqsubs=18, package="systemPipeR"))
```

5.7.2 Alignment with BWA-MEM (e.g. for VAR-Seq)

The following example runs BWA-MEM as a single process without submitting it to a cluster.

```
args <- systemArgs(sysma="bwa.param", mytargets="targets.txt")
moduleload(modules(args)) # Skip if module system is not available
system("bwa index -a bwtsw ./data/tair10.fasta") # Indexes reference genome
bampaths <- runCommandline(args=args)
```

5.7.3 Alignment with Rsubread (e.g. for RNA-Seq)

The following example shows how one can use within the *systemPipeR* environment the R-based aligner *Rsubread* or other R-based functions that read from input files and write to output files.

```
library(Rsubread)
args <- systemArgs(sysma="rsubread.param", mytargets="targets.txt")
buildindex(basename=reference(args), reference=reference(args)) # Build indexed reference genome
align(index=reference(args), readfile1=infile1(args), input_format="FASTQ",
      output_file=outfile1(args), output_format="SAM", nthreads=8, indels=1, TH1=2)
for(i in seq(along=outfile1(args))) asBam(file=outfile1(args)[i], destination=gsub(".sam", "", outfile1(ar
```

5.7.4 Alignment with `gsnap`

Another R-based short read aligner is gsnap from the *gmapR* package (Wu and Nacu, 2010). The code sample below introduces how to run this aligner on multiple nodes of a compute cluster.

```
library(gmapR); library(BiocParallel); library(BatchJobs)
gmapGenome <- GmapGenome(reference(args), directory="data", name="gmap_tair10chr/", create=TRUE)
args <- systemArgs(sysma="gsnap.param", mytargets="targetsPE.txt")
f <- function(x) {
    library(gmapR); library(systemPipeR)
    args <- systemArgs(sysma="gsnap.param", mytargets="targetsPE.txt")
    gmapGenome <- GmapGenome(reference(args), directory="data", name="gmap_tair10chr/", create=FALSE)
    p <- GsnapParam(genome=gmapGenome, unique_only=TRUE, molecule="DNA", max_mismatches=3)
    o <- gsnap(input_a=infile1(args)[x], input_b=infile2(args)[x], params=p, output=outfile1(args)[x])
}
funs <- makeClusterFunctionsTorque("torque.tmpl")
param <- BatchJobsParam(length(args), resources=list(walltime="20:00:00", nodes="1:ppn=1", memory="6gb"), 
register(param)
d <- bplapply(seq(along=args), f)
```

5.8 Read counting for mRNA profiling experiments

Create `txdb` (needs to be done only once)

```
library(GenomicFeatures)
txdb <- makeTxDbFromGFF(file="data/tair10.gff", format="gff", dataSource="TAIR", organism="A. thaliana")
saveDb(txdb, file="./data/tair10.sqlite")
```

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

```
library(BiocParallel)
txdb <- loadDb("./data/tair10.sqlite")
```

```
eByg <- exonsBy(txdb, by="gene")
bfl <- BamFileList(outpaths(args), yieldSize=50000, index=character())
multicoreParam <- MulticoreParam(workers=4); register(multicoreParam); registered()
counteByg <- bplapply(bfl, function(x) summarizeOverlaps(eByg, x, mode="Union", ignore.strand=TRUE, inter.f
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")
```

Read counting with `summarizeOverlaps` using multiple nodes of a cluster

```
library(BiocParallel)
f <- function(x) {
    library(systemPipeR); library(BiocParallel); library(GenomicFeatures)
    txdb <- loadDb("./data/tair10.sqlite")
    eByg <- exonsBy(txdb, by="gene")
    args <- systemArgs(sysma="tophat.param", mytargets="targets.txt")
    bfl <- BamFileList(outpaths(args), yieldSize=50000, index=character())
    summarizeOverlaps(eByg, bfl[x], mode="Union", ignore.strand=TRUE, inter.feature=TRUE, singleEnd=TRUE)
}
funs <- makeClusterFunctionsTorque("torque.tmpl")
param <- BatchJobsParam(length(args), resources=list(walltime="20:00:00", nodes="1:ppn=1", memory="6gb"), c
register(param)
counteByg <- bplapply(seq(along=args), f)
countDFeByg <- sapply(seq(along=counteByg), function(x) assays(counteByg[[x]])$counts)
rownames(countDFeByg) <- names(rowRanges(counteByg[[1]])); colnames(countDFeByg) <- names(outpaths(args))
```

5.9 Read counting for miRNA profiling experiments

Download miRNA genes from miRBase

```
system("wget ftp://mirbase.org/pub/mirbase/19/genomes/My_species.gff3 -P ./data/")
gff <- import.gff("./data/My_species.gff3", asRangedData=FALSE)
gff <- split(gff, elementMetadata(gff)$ID)
bams <- names(bampaths); names(bams) <- targets$SampleName
bfl <- BamFileList(bams, yieldSize=50000, index=character())
countDFmiR <- summarizeOverlaps(gff, bfl, mode="Union", ignore.strand=FALSE, inter.feature=FALSE) # Note:
rpkmDFmiR <- apply(countDFmiR, 2, function(x) returnRPKM(counts=x, gffsub=gff))
write.table(assays(countDFmiR)$counts, "results/countDFmiR.xls", col.names=NA, quote=FALSE, sep="\t")
write.table(rpkmDFmiR, "results/rpkmDFmiR.xls", col.names=NA, quote=FALSE, sep="\t")
```

5.10 Correlation analysis of samples

The following computes the sample-wise Spearman correlation coefficients from the RPKM normalized expression values. 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).

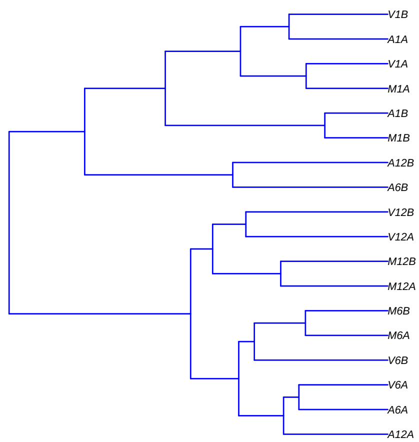

Figure 2: Correlation dendrogram of samples.

5.11 DEG analysis with edgeR

The following `run_edgeR` function is a convenience wrapper for identifying differentially expressed genes (DEGs) in batch mode with *edgeR*'s GML method (Robinson et al., 2010) for any number of pairwise sample comparisons specified under the `cmp` argument. Users are strongly encouraged to consult the *edgeR* vignette for more detailed information on this topic and how to properly run *edgeR* on data sets with more complex experimental designs.

```
targets <- read.delim(targetspath, comment="#")
cmp <- readComp(file=targetspath, format="matrix", delim="-")
cmp[[1]]

      [,1]  [,2]
 [1,] "M1"  "A1"
 [2,] "M1"  "V1"
 [3,] "A1"  "V1"
 [4,] "M6"  "A6"
 [5,] "M6"  "V6"
 [6,] "A6"  "V6"
 [7,] "M12" "A12"
 [8,] "M12" "V12"
 [9,] "A12" "V12"
```

```
countDFeBygpath <- system.file("extdata", "countDFeByg.xls", package="systemPipeR")
countDFeByg <- read.delim(countDFeBygpath, row.names=1)
edgeDF <- run_edgeR(countDF=countDFeByg, targets=targets, cmp=cmp[[1]], independent=FALSE, mdsplot="")

Disp = 0.20653 , BCV = 0.4545
```

Filter and plot DEG results for up and down regulated genes. Because of the small size of the toy data set used by this vignette, the `FDR` value has been set to a relatively high threshold (here 10%). More commonly used `FDR` cutoffs are 1% or 5%.

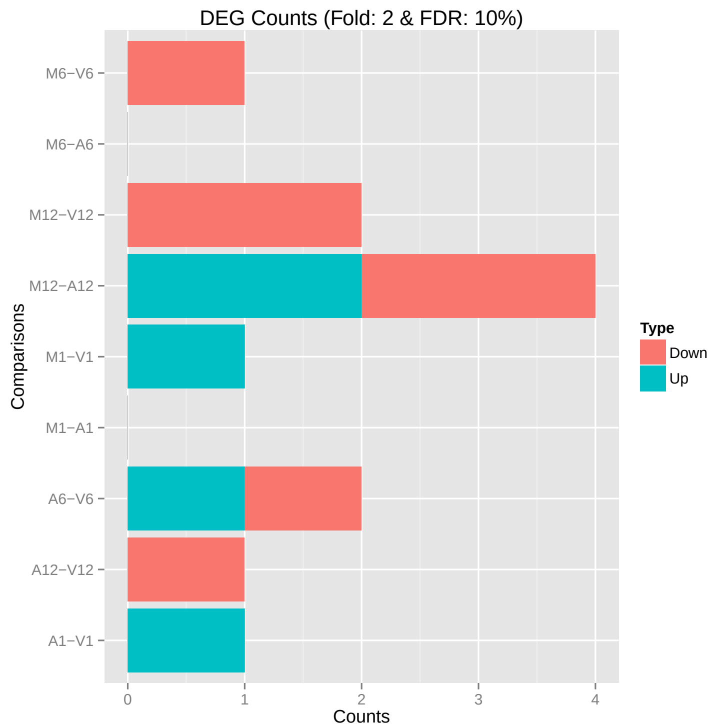

Figure 3: Up and down regulated DEGs identified with *edgeR*.

```
names(DEG_list)

[1] "UporDown" "Up"       "Down"     "Summary"

DEG_list$Summary[1:4,]

      Comparisons Counts_Up_or_Down Counts_Up Counts_Down
M1-A1       M1-A1                 0         0           0
M1-V1       M1-V1                 1         1           0
A1-V1       A1-V1                 1         1           0
M6-A6       M6-A6                 0         0           0
```

5.12 DEG analysis with DESeq2

The following run_DESeq2 function is a convenience wrapper for identifying DEGs in batch mode with *DESeq2* (Love et al., 2014) for any number of pairwise sample comparisons specified under the cmp argument. Users are strongly encouraged to consult the *DESeq2* vignette for more detailed information on this topic and how to properly run *DESeq2* on data sets with more complex experimental designs.

```
degseqDF <- run_DESeq2(countDF=countDFeByg, targets=targets, cmp=cmp[[1]], independent=FALSE)
```

Filter and plot DEG results for up and down regulated genes.

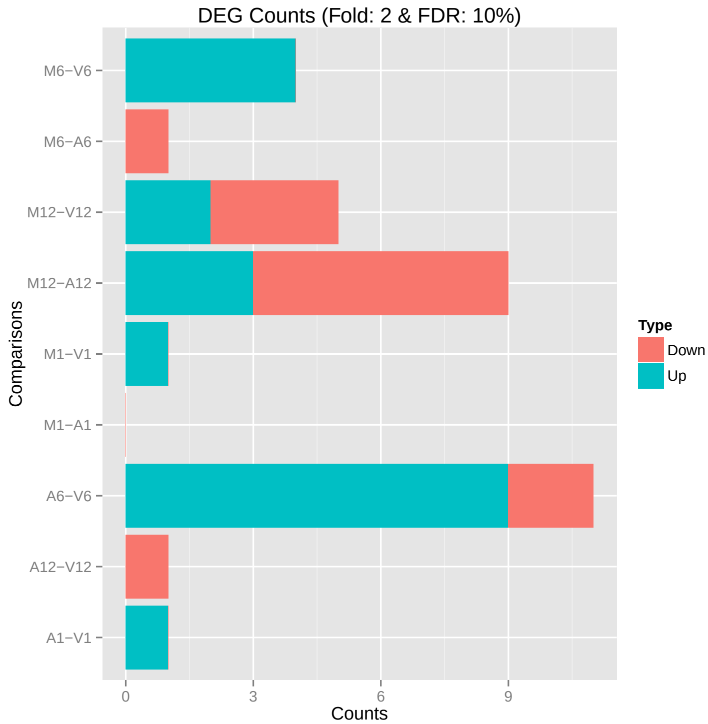

Figure 4: Up and down regulated DEGs identified with *DESeq2*.

5.13 Venn Diagrams

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).

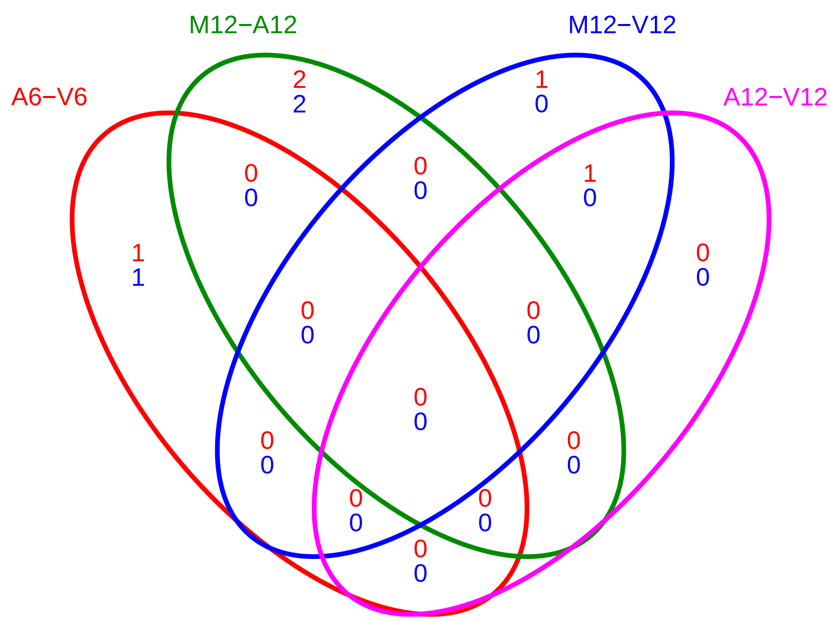

Figure 5: Venn Diagram for 4 Up and Down DEG Sets.

5.14 GO term enrichment analysis of DEGs

5.14.1 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])
```

```
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")
```

5.14.2 Batch GO term enrichment analysis

Apply the enrichment analysis to the DEG sets obtained in 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 one can obtain such a GO slim vector from BioMart for a specific organism.

```
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
```

5.14.3 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")
```

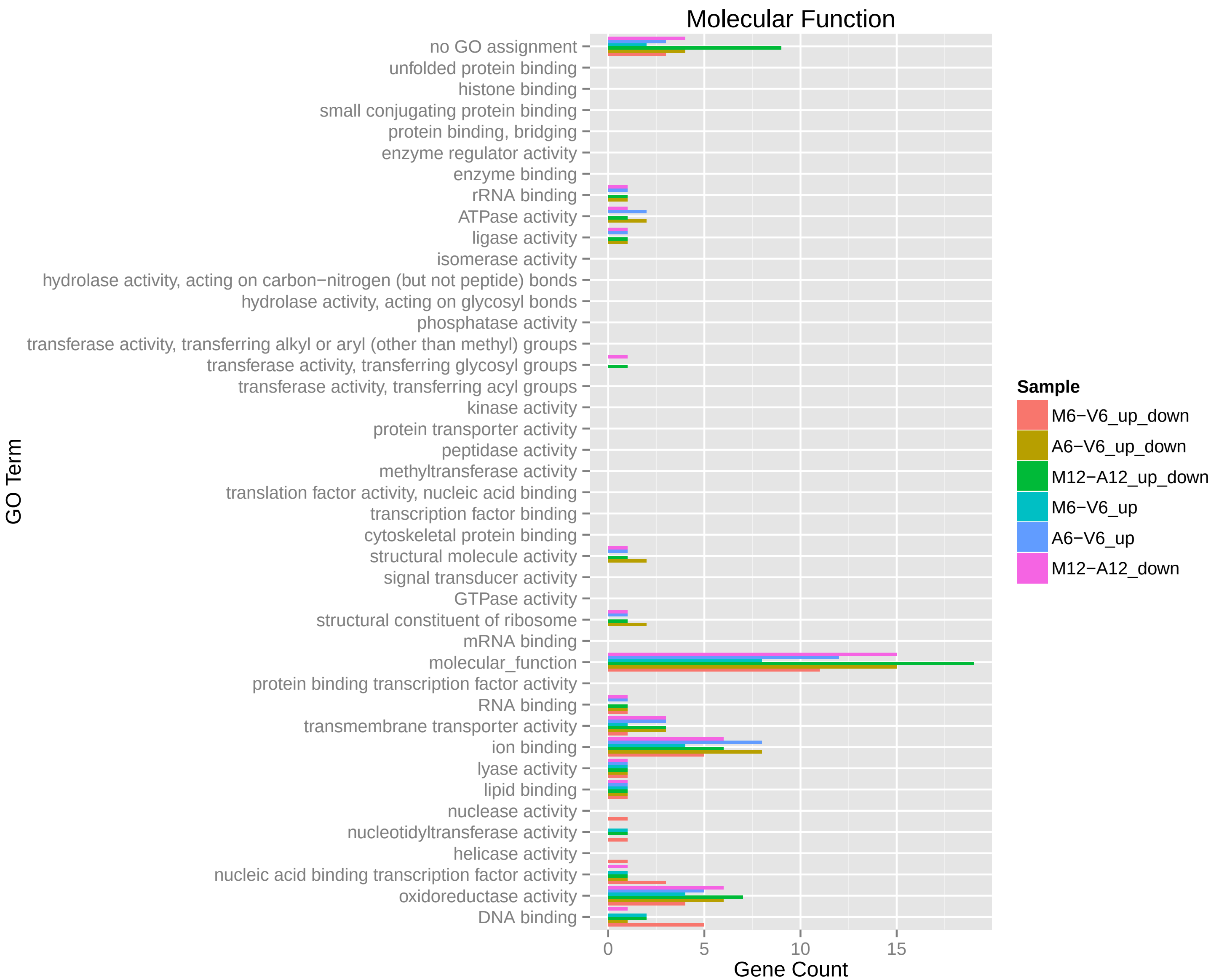

Figure 6: GO Slim Barplot for MF Ontology.

5.15 Clustering and heat maps

The following example performs hierarchical clustering on the RPKM normalized 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 <- rpkmDFeByg[geneids, ]
pdf("heatmap1.pdf")
pheatmap(y, scale="row", clustering_distance_rows="correlation", clustering_distance_cols="correlation")
dev.off()
```

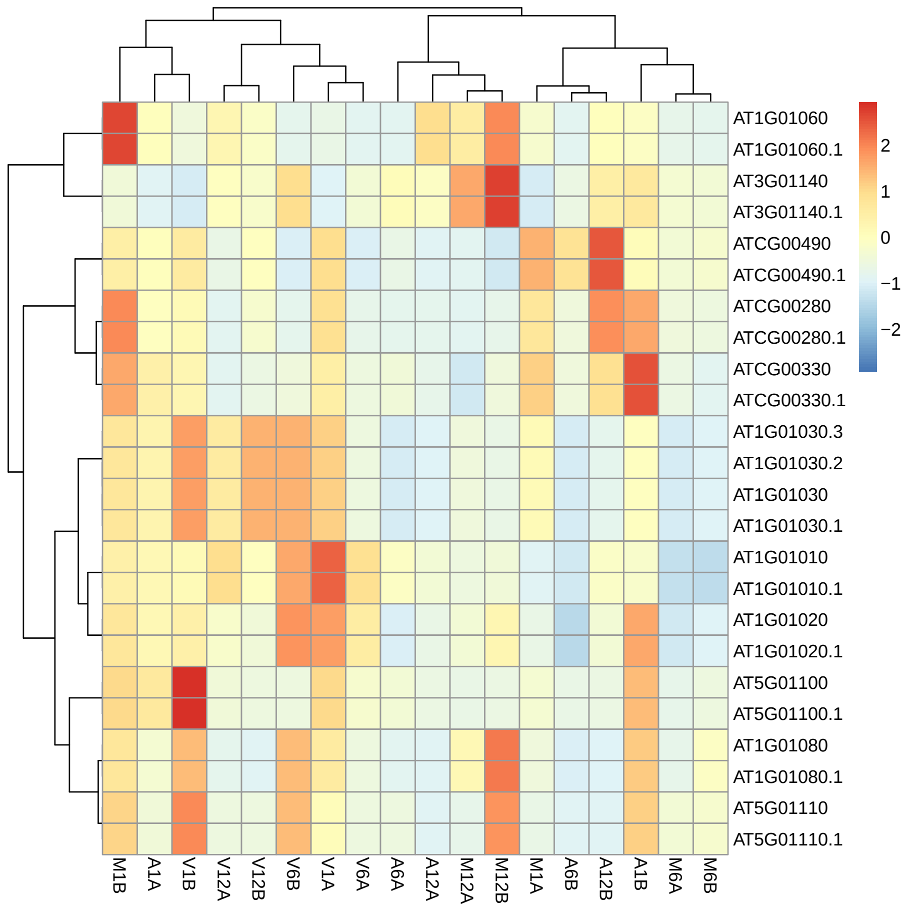

Figure 7: Heat map with hierarchical clustering dendrograms of DEGs.

6 Version Information

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

- R version 3.2.1 (2015-06-18), `x86_64-unknown-linux-gnu`
- Locale: `C`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: Biobase 2.29.1, BiocGenerics 0.15.3, BiocParallel 1.3.31, Biostrings 2.37.2, DBI 0.3.1, GenomeInfoDb 1.5.8, GenomicAlignments 1.5.11, GenomicRanges 1.21.16, IRanges 2.3.14, RSQLite 1.0.0, Rsamtools 1.21.12, S4Vectors 0.7.10, ShortRead 1.27.5, SummarizedExperiment 0.3.2, XVector 0.9.1, ape 3.3, knitr 1.10.5, systemPipeR 1.3.15
- Loaded via a namespace (and not attached): AnnotationDbi 1.31.17, AnnotationForge 1.11.7, BBmisc 1.9, BatchJobs 1.6, BiocStyle 1.7.4, Category 2.35.1, DESeq2 1.9.15, Formula 1.2-1, GO.db 3.1.2, GOstats 2.35.1, GSEABase 1.31.3, Hmisc 3.16-0, MASS 7.3-42, Matrix 1.2-1, RBGL 1.45.1, RColorBrewer 1.1-2, Rcpp 0.11.6, RcppArmadillo 0.5.200.1.0, XML 3.98-1.3, acepack 1.3-3.3, annotate 1.47.0, base64enc 0.1-2, bitops 1.0-6, brew 1.0-6, checkmate 1.6.0, cluster 2.0.2, colorspace 1.2-6, digest 0.6.8, edgeR 3.11.2, evaluate 0.7, fail 1.2, foreign 0.8-65, formatR 1.2, futile.logger 1.4.1, futile.options 1.0.0, genefilter 1.51.0, geneplotter 1.47.0, ggplot2 1.0.1, graph 1.47.2, grid 3.2.1, gridExtra 0.9.1, gtable 0.1.2, highr 0.5, hwriter 1.3.2, labeling 0.3, lambda.r 1.1.7, lattice 0.20-31, latticeExtra 0.6-26, limma 3.25.12, locfit 1.5-9.1, magrittr 1.5, munsell 0.4.2,

nlme 3.1-121, nnet 7.3-10, pheatmap 1.0.7, plyr 1.8.3, proto 0.3-10, reshape2 1.4.1, rjson 0.2.15, rpart 4.1-10, scales 0.2.5, sendmailR 1.2-1, splines 3.2.1, stringi 0.5-5, stringr 1.0.0, survival 2.38-3, tools 3.2.1, xtable 1.7-4, zlibbioc 1.15.0

7 Funding

This software was developed with funding from the Agriculture and Food Research Institute of the National Institute of Food and Agriculture of the USDA (2011-68004-30154), the National Science Foundation (MCB-1021969) and the National Institutes of Health/National Institute of Allergy and Infectious Diseases (5R01 AI036959).

8 References

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

Brian E Howard, Qiwen Hu, Ahmet Can Babaoglu, Manan Chandra, Monica Borghi, Xiaoping Tan, Luyan He, Heike Winter-Sederoff, Walter Gassmann, Paola Veronese, and Steffen Heber. High-throughput RNA sequencing of pseudomonas-infected arabidopsis reveals hidden transcriptome complexity and novel splice variants. *PLoS One*, 8 (10):e74183, 1 October 2013. ISSN 1932-6203. doi: 10.1371/journal.pone.0074183. URL http://dx.doi.org/10.1371/journal.pone.0074183.

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.

H Li and R Durbin. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics*, 25(14): 1754–1760, July 2009. ISSN 1367-4803. doi: 10.1093/bioinformatics/btp324. URL http://dx.doi.org/10.1093/bioinformatics/btp324.

Heng Li. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. 03 2013. URL http://arxiv.org/abs/1303.3997.

Yang Liao, Gordon K Smyth, and Wei Shi. The subread aligner: fast, accurate and scalable read mapping by seed-and-vote. *Nucleic Acids Res.*, 41(10):e108, 4 April 2013. ISSN 0305-1048. doi: 10.1093/nar/gkt214. URL http://dx.doi.org/10.1093/nar/gkt214.

Michael Love, Wolfgang Huber, and Simon Anders. Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. *Genome Biol.*, 15(12):550, 2014. ISSN 1465-6906. doi: 10.1186/s13059-014-0550-8. URL http://genomebiology.com/2014/15/12/550.

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.

T D Wu and S Nacu. Fast and SNP-tolerant detection of complex variants and splicing in short reads. *Bioinformatics*, 26(7):873–881, April 2010. ISSN 1367-4803. doi: 10.1093/bioinformatics/btq057. URL http://dx.doi.org/10.1093/bioinformatics/btq057.