Analysis of RNA-Seq Data with R/Bioconductor

...

Thomas Girke

December 6, 2014

Overview

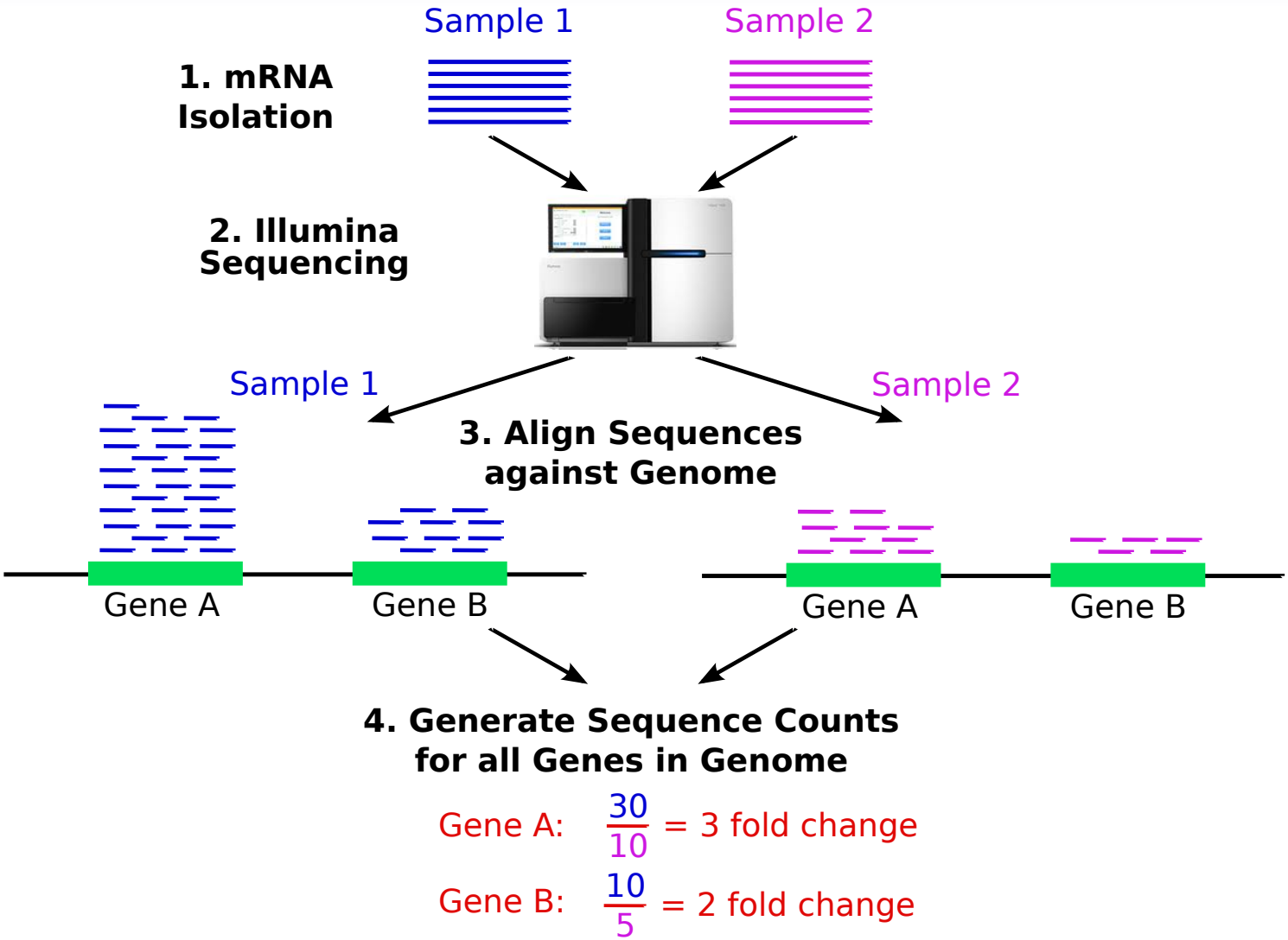
RNA-Seq Analysis

- Quality Report
- Aligning Short Reads
- Counting Reads per Feature
- DEG Analysis
- GO Analysis
- View Results in IGV & ggbio
- Differential Exon Usage

References

Outline

RNA-Seq Technology



Analysis Workflow of RNA-Seq Gene Expression Data

1. Alignment of RNA reads to reference
   - Reference can be genome or transcriptome.
2. Count reads overlapping with annotation features of interest
   - Most common: counts for exonic gene regions, but many viable alternatives exist here: counts per exons, genes, introns, etc.
3. Normalization
   - Main adjustment for sequencing depth and compositional bias.
4. Identification of Differentially Expressed Genes (DEGs)
   - Identification of genes with significant expression differences.
   - Identification of expressed genes possible for strongly expressed ones.
5. Specialty applications
   - Splice variant discovery (semi-quantitative), gene discovery, antisense expressions, etc.
6. Cluster Analysis
   - Identification of genes with similar expression profiles across many samples.
7. Enrichment Analysis of Functional Annotations
   - Gene ontology analysis of obtained gene sets from steps 5-6.

Important Aspects in RNA-Seq Analysis

- Alignment reference
  - Genome
  - Transcript models
  - Both
- How to quantify expression?
  - Read count per range
  - Coverage statistics per range
- What features?
  - Genes, transcript models, exons
- Alternative splicing
  - Often restricted to splice junction analysis
  - Objective: discovery vs. quantification

Important Considerations for NGS Alignments

- In NGS we usually want to find the origin of reads (NG sequences) in a reference genome or transcriptome. Thus, we are mostly interested in finding the best scoring or multiple best scoring locations for each read, but not lower scoring alternative solutions as in paralog/ortholog search applications.
- Ambiguous mappings should be removed, because there is no evidence for their origin. However, for certain applications one needs to include them, *e.g.* when mapping RNA-Seq reads against transcript sequences instead of genome.

Short Read Aligner for RNA-Seq

No special requirements for alignments with low number of variants

- ChIP-Seq
- RNA-Seq (if mapping against transcriptome or intron-less genome)
- Bis-Seq (with injected reference)
- ...

Variant tolerant aligners to account for mismatches and indels

- VAR-Seq
- Bis-Seq (without injected reference)
- ...

Splice tolerant aligner to account for introns

- RNA-Seq (if mapping against genome with introns)

Sequence Alignment/Map (SAM/BAM) Format

SAM is a tab-delimited alignment format consisting of a header section (lines starting with `@`) and an alignment section with 12 columns. BAM is the compressed, indexed and binary version of this format.

The below sample alignment contains the following features: (1) bases in lower cases are clipped from the alignment; (2) read r001/1 and r001/2 constitute a read pair; (3) r003 is a chimeric read; (4) r004 represents a split alignment.

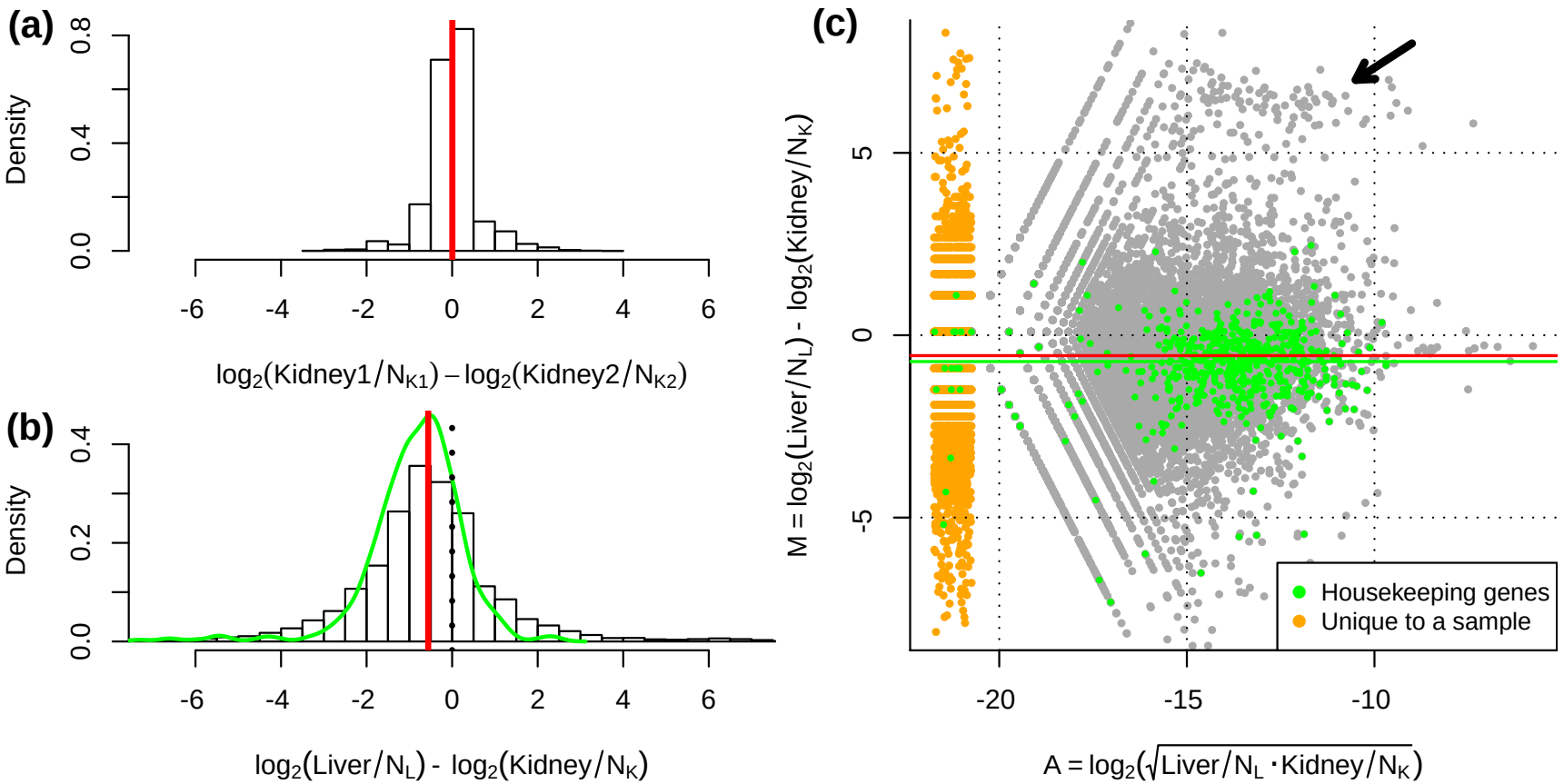
```
Coor     12345678901234  5678901234567890123456789012345
ref      AGCATGTTAGATAA**GATAGCTGTGCTAGTAGGCAGTCAGCGCCAT
+r001/1        TTAGATAAAGGATA*CTG
+r002         aaaAGATAA*GGATA
+r003       gcctaAGCTAA
+r004                     ATAGCT..............TCAGC
-r003                           ttagctTAGGC
-r001/2                                        CAGCGGCAT
```

⇓ SAM Format

```
r001  163 ref  7 30 8M2I4M1D3M = 37  39 TTAGATAAAGGATACTG   *
r002    0 ref  9 30 3S6M1P1I4M *  0   0 AAAAGATAAGGATA      *
r003    0 ref  9 30 5S6M       *  0   0 GCCTAAGCTAA         * SA:Z:ref,29,-,6H5M,17,0;
r004    0 ref 16 30 6M14N5M    *  0   0 ATAGCTTCAGC         *
r003 2064 ref 29 17 6H5M       *  0   0 TAGGC               * SA:Z:ref,9,+,5S6M,30,1;
r001   83 ref 37 30 9M         =  7 -39 CAGCGGCAT           * NM:i:1
```

For details see the SAM Format Specification Link

Normalization Required



Log ratio distributions (a and b) and MA plot (c) for two tissue samples (from Robinson and Oshlack, 2010).

Be Careful with RPKM/FPKM Values

RPKM Concept (FPKM is paired-end version of it)

- RPKM (FPKM): reads (fragments) per kp per million mapped reads
- The more we sequence, the more reads we expect from each gene. This is the most relevant correction of this method.
- Longer transcript are expected to generate more reads. The latter is only relevant for comparisons among different genes which we rarely perform!
- RPKM/FPKM are not suitable for statistical testing. Why? Consider the following example: in two libraries, each with one million reads, gene X may have 10 reads for treatment A and 5 reads for treatment B, while it is 100x as many after sequencing 100 millions reads from each library. In the latter case we can be much more confident that there is a true difference between the two treatments than in the first one. However, the RPKM values would be the same for both scenarios.
- Thus, RPKM/FPKM are useful for reporting expression values, but not for statistical testing!

TMM Method Corrects for RNA Composition Bias

Trimmed Mean of M Values (TMM) by Robinson and Oshlack (2010)

- Many normalization RNA-Seq normalization methods perform poorly on samples with extreme composition bias. For instance, in one sample a large number of reads comes from rRNAs while in another they have been removed more efficiently. Most scaling based methods, including RPKM and CPM, will underestimate the expression of weaker expressed genes in the presence of extremely abundant mRNAs (less sequencing real estate available for them). The TMM methods tries to correct this bias.
- Method implemented in edgeR library (Robinson et al., 2010).

Analysis of Differentially Expressed Genes (DEGs)

- Data is discrete, positively skewed
  $\Rightarrow$ no (log-)normal model
- Small numbers of replicates
  $\Rightarrow$ no rank based or permutation methods
- Sequencing depth (coverage) varies among samples
  $\Rightarrow$ normalization

DEG Analysis Methods

Requirements

- One would like to perform a t-test or something similar for each gene.
- t-test assumes normal distribution and no mean-variance dependence. Both are not appropriate assumptions for RNA-Seq data.
- Variance estimation and rank-order statistics is difficult on small sample numbers.

Statistical Testing

- Poisson distribution (initially used but not very common anymore)
- Most statistical methods for RNA-Seq DEG analysis use negative binomial distribution along with modified statistical tests based on that.
- The mutiple testing issue is very similar as in microarray data analysis. Thus, most tools provide False Discovery Rates (FDRs), which are derived from p-values corrected for multiple testing using the Benjamini-Hochberg method.
- For variance estimation most methods borrow information across genes

Software for RNA-Seq DEG Analysis

- edgeR (Robinson et al., 2010)
- DESeq/DESeq2 (Anders and Huber, 2010)
- DEXSeq (Anders et al., 2012)
- limmaVoom
- Cuffdiff/Cuffdiff2 (Trapnell et al., 2013)
- PoissonSeq
- baySeq
- ...

Packages for RNA-Seq Analysis in R

- GenomicRanges Link: high-level infrastructure for range data
- Rsamtools Link: BAM support
- rtracklayer Link: Import/export of range and annotation data, interface to online genome browsers, etc.
- DESeq Link: RNA-Seq DEG analysis
- DESeq2 Link: RNA-Seq DEG analysis
- edgeR Link: RNA-Seq DEG analysis
- DEXSeq Link: RNA-Seq Exon analysis
- QuasR Link: RNA-Seq workflows
- systemPipeR Link: NGS workflows and reports

Outline

Data Sets and Experimental Variables

To make the following sample code work, please follow these instructions:

- Download and unpack the sample data Link for this practical.
- Direct your R session to the resulting `Rrnaseq` directory. It contains 18 slimmed down FASTQ files (SRP010938 Link) from *A. thaliana* (Howard et al., 2013). To minimize processing time, each FASTQ file has been subsetted to 90,000-100,000 randomly sampled reads that map to the first 100,000 nucleotides of each chromosome. The corresponding reference genome sequence (FASTA) and its GFF annotion files have been truncated accordingly.
- Start the analysis by opening in your R session the `Rrnaseq.R` script Link which contains the code shown in this slide show in pure text format.

The FASTQ files are organized in the provided `targets.txt` file Link. This is the only file in this analysis workflow that needs to be generated manually, *e.g.* in a spreadsheet program. To import `targets.txt`, we run the following commands from R:

```
> # download.file("http://biocluster.ucr.edu/~tgirke/HTML_Presentations/Manuals/Workshop_Dec_5_8_2014/Rrnaseq.z
> # unzip("Rrnaseq.zip")

> library(systemPipeR)
> args <- systemArgs(sysma="tophat.param", mytargets="targets.txt")
> targets <- read.delim("targets.txt", comment.char = "#")
> targets[1:3,]
```

```
                 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
```

Outline

Quality Reports

The following shows how to create read quality reports with the `seeFastq` function from *systemPipeR*.

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

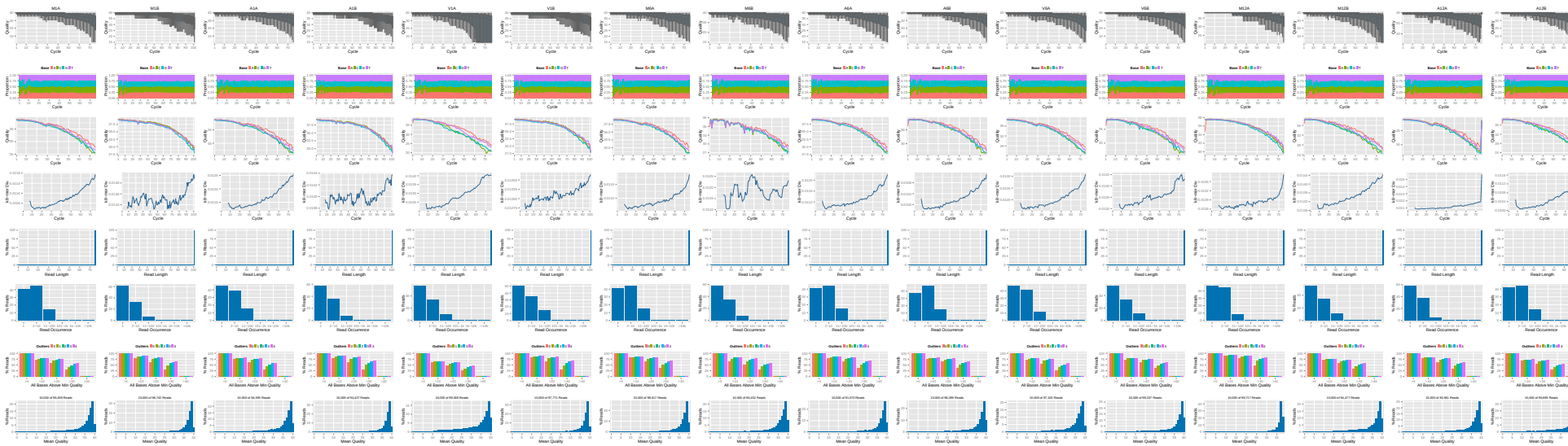


Figure: QC report for 18 FASTQ files.

Outline

Align Reads Option 1: *QuasR*

QuasR is an extremely versatile NGS mapping and postprocessing pipeline for RNA-Seq and many other application areas, such as BS-Seq, allele-specific RNA-Seq, etc. It uses *Rbowtie* for ungapped alignments and SpliceMap for spliced alignments.

(1) Evironment settings

```
> library(QuasR)
> QuasR_samples <- targets[,1:2]; QuasR_samples[,1] <- gsub(".*/", "", QuasR_samples[,1])
> write.table(QuasR_samples, "data/QuasR_samples.txt", row.names=FALSE, quote=FALSE, sep="\t")
> sampleFile <- "./data/QuasR_samples.txt"
> genomeFile <- "./data/tair10.fasta"
> results <- "./results" # defines location where to write results
> cl <- makeCluster(1) # defines number of CPU cores to use
```

(2) Single command to index reference, align all samples and generate BAM files.

```
> proj <- qAlign(sampleFile, genome=genomeFile, maxHits=1, splicedAlignment=FALSE, alignmentsDir=results,
+                clObj=cl, cacheDir=results)
>                # Note: splicedAlignment should be set to TRUE when the reads are >=50nt long
> (alignstats <- alignmentStats(proj)[1:4,]) # Alignment summary report
```

Align Reads Option 2: *Rsubread*

Rsubread is an R/Bioc package that implements an extremely fast aligner for RNA-Seq data. It is currently only available for OS X and Linux, but not for Windows.

(1) Index reference genome

```
> library(Rsubread); library(systemPipeR)
> args <- systemArgs(sysma="rsubread.param", mytargets="targets.txt")
> buildindex(basename=reference(args), reference=reference(args)) # Build indexed re
```

(2) Align all FASTQ files with *Rsubread* in loop. Includes generation of indexed BAM files.

```
> 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
> unlink(outfile1(args)); unlink(paste0(outfile1(args), ".indel"))
```

Align Reads Option 3: Tophat2 using *systemPipeR*

systemPipeR NGS workflow and report generation environment that can run command-line software on local computers and compute clusters. Note: this step requires the command-line tools `tophat2/bowtie2` Link.

(1) Index reference genome

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

(2) Align all FASTQ files with Bowtie2/Tophat2 on a single computer. Includes generation of indexed BAM files.

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

```
Missing alignment results (bam files): 0
Existing alignment results (bam files): 18
```

```
> bampaths
```

```
   M1A    M1B    A1A    A1B    V1A    V1B    M6A    M6B    A6A    A6B    V6A    V6B   M12A   M12B   A12A   A12
"TRUE" "TRUE" "TRUE" "TRUE" "TRUE" "TRUE" "TRUE" "TRUE" "TRUE" "TRUE" "TRUE" "TRUE" "TRUE" "TRUE" "TRUE" "TRUE"
```

(3) Alternatively, align all FASTQ files with Bowtie2/Tophat2 on many compute nodes in parallel. The following submits to Torque scheduler 18 processes each with 4 CPU cores.

```
> 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=r
> showStatus(reg)
```

Alignment Summary

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))[1:8,]
```

	FileName	Nreads	Nalign	Perc_Aligned	Nalign_Primary	Perc_Aligned_Primary
M1A	M1A	96459	89376	92.65698	89376	92.65698
M1B	M1B	98742	86014	87.10984	86014	87.10984
A1A	A1A	94935	88360	93.07421	88360	93.07421
A1B	A1B	94427	83172	88.08074	83172	88.08074
V1A	V1A	99366	80869	81.38498	80869	81.38498
V1B	V1B	97771	93637	95.77175	93637	95.77175
M6A	M6A	98617	92917	94.22006	92917	94.22006
M6B	M6B	90452	80074	88.52651	80074	88.52651

```
> write.table(read_statsDF, "results/alignStats.xls", row.names=FALSE, quote=FALSE,
```

Outline

Import Annotation Data from GFF

Annotation data from GFF

```
> library(rtracklayer); library(GenomicRanges); library(Rsamtools)
> gff <- import.gff("./data/tair10.gff", asRangedData=FALSE)
> seqlengths(gff) <- end(ranges(gff[which(elementMetadata(gff)[,"type"]=="chromosome
> subgene_index <- which(elementMetadata(gff)[,"type"] == "exon")
> gffsub <- gff[subgene_index,] # Returns only gene ranges
> gffsub[1:4, c(2,5)]

GRanges object with 4 ranges and 2 metadata columns:
      seqnames       ranges strand |     type              group
         <Rle>    <IRanges>  <Rle> | <factor>           <factor>
  [1]     Chr1 [3631, 3913]      + |     exon Parent=AT1G01010.1
  [2]     Chr1 [3996, 4276]      + |     exon Parent=AT1G01010.1
  [3]     Chr1 [4486, 4605]      + |     exon Parent=AT1G01010.1
  [4]     Chr1 [4706, 5095]      + |     exon Parent=AT1G01010.1
  -------
  seqinfo: 7 sequences from an unspecified genome

> ids <- gsub("Parent=|\\..*", "", elementMetadata(gffsub)$group)
> gffsub <- split(gffsub, ids) # Coerce to GRangesList
```

More Robust: Store Annotations in TranscriptDb

Storing annotation ranges in *TranscriptDb* databases makes many operations more robust and convenient.

```
> library(GenomicFeatures)
> txdb <- makeTranscriptDbFromGFF(file="data/tair10.gff",
+         format="gff3",
+         dataSource="TAIR",
+         species="Arabidopsis thaliana")
> saveDb(txdb, file="./data/tair10.sqlite")
> txdb <- loadDb("./data/tair10.sqlite")
> eByg <- exonsBy(txdb, by="gene")
```

Old Read Counting with countOverlaps

Number of reads overlapping gene ranges

```
> args <- systemArgs(sysma="tophat.param", mytargets="targets.txt")
> countDF <- data.frame(row.names=names(eByg))
> for(i in outpaths(args)) {
+         aligns <- readGAlignmentsFromBam(i) # Substitute next two lines with this
+         counts <- countOverlaps(eByg, aligns, ignore.strand=TRUE)
+         countDF <- cbind(countDF, counts)
+ }
> colnames(countDF) <- names(outpaths(args))
> countDF[1:4,]
```

	M1A	M1B	A1A	A1B	V1A	V1B	M6A	M6B	A6A	A6B	V6A	V6B	M12A	M12B	A12A	A12B	V12A
AT1G01010	28	128	99	87	183	118	22	19	77	23	148	201	60	68	66	33	113
AT1G01020	12	47	35	50	49	41	9	11	9	3	42	71	21	37	16	9	20
AT1G01030	19	51	36	33	47	78	5	8	4	4	18	73	16	15	7	5	34
AT1G01040	98	354	259	345	298	350	82	81	128	34	359	530	171	298	183	41	101

New Read Counting with summarizeOverlaps

The summarizeOverlaps function from the GenomicRanges package is easier to use, it provides more options and it is much more memory efficient. See here Link for details.

```
> 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=2); register(multicoreParam)
> counteByg <- bplapply(bfl, function(x) summarizeOverlaps(eByg, x, mode="Union", ig
> countDFeByg <- sapply(seq(along=counteByg), function(x) assays(counteByg[[x]])$cou
> rownames(countDFeByg) <- names(rowData(counteByg[[1]])); colnames(countDFeByg) <-
> countDFeByg[1:4,1:12]
```

	M1A	M1B	A1A	A1B	V1A	V1B	M6A	M6B	A6A	A6B	V6A	V6B
AT1G01010	28	128	99	87	183	118	22	19	77	23	148	201
AT1G01020	12	47	35	50	49	41	9	11	9	3	42	71
AT1G01030	19	51	36	33	47	78	5	8	4	4	18	73
AT1G01040	98	354	259	345	298	350	82	81	128	34	359	530

```
> write.table(countDFeByg, "results/countDFeByg.xls", col.names=NA, quote=FALSE, sep
```

Simple RPKM Normalization

RPKM: reads per kilobase of exon model per million mapped reads

```
> rpkmDFeByg <- apply(countDFeByg, 2, function(x) returnRPKM(counts=x, ranges=eByg))
> write.table(rpkmDFeByg, "results/rpkmDFeByg.xls", col.names=NA, quote=FALSE, sep="
> rpkmDFeByg[1:4,1:7]
```

	M1A	M1B	A1A	A1B	V1A	V1B	M6A
AT1G01010	2424.745	6201.798	5431.999	4225.653	11608.558	5482.332	1315.5522
AT1G01020	988.799	2166.827	1827.306	2310.805	2957.618	1812.533	512.0905
AT1G01030	1457.938	2189.552	1750.267	1420.254	2641.816	3211.111	264.9311
AT1G01040	2290.597	4629.408	3835.648	4522.808	5102.206	4389.003	1323.4691

Read Counting with qCount from *QuasR*

QuasR does everything in one command.

```
> countDF3 <- qCount(proj, txdb, reportLevel="gene", orientation="any")
> countDF3[1:4,1:12]
> write.table(countDF3, "results/countDFgene.xls", col.names=NA, quote=FALSE, sep="
```

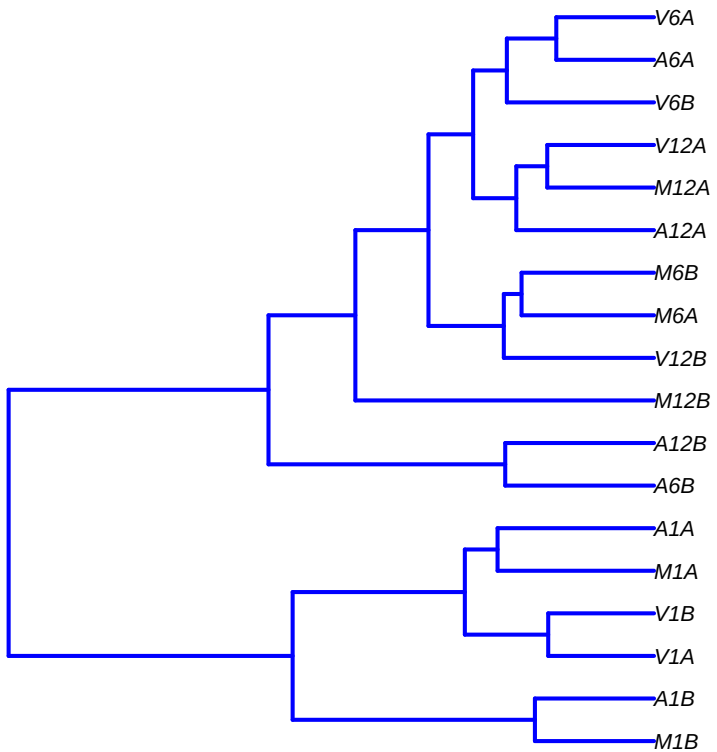
RPKM: for QuasR results

```
> rpkmDFgene <- t(t(countDF3[,-1]/countDF3[,1] * 1000)/colSums(countDF3[,-1]) *1e6)
```

Reproducibility Check by Sample-Wise Clustering

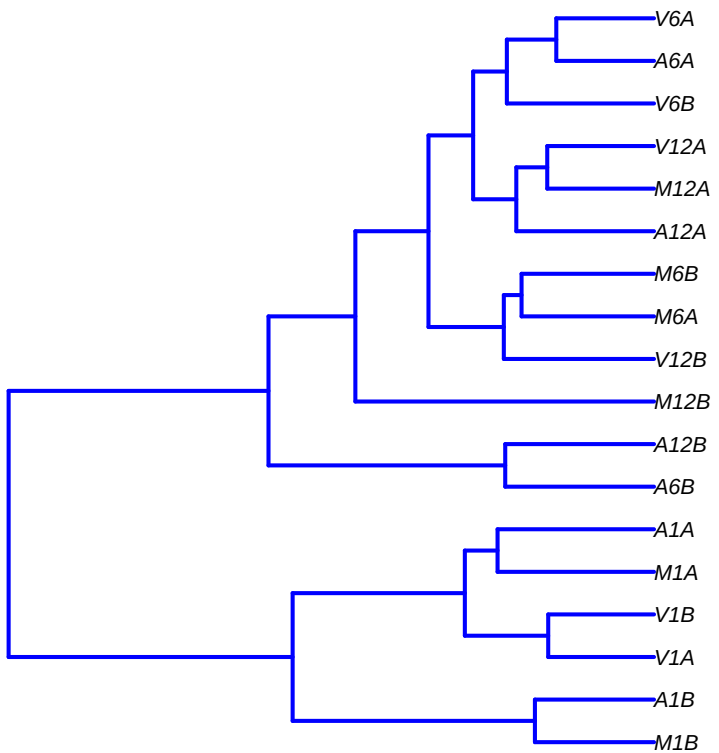
QC check of the sample reproducibility by computing a correlating matrix and plotting it as a tree. Note: the `plotMDS` function from *edgeR* is a more robust method for this task.

```
> library(ape)
> d <- cor(rpkmDFeByg, method="spearman")
> hc <- hclust(dist(1-d))
> plot.phylo(as.phylo(hc), type="p", edge.col=4, edge.width=3, show.node.label=TRUE, no.margin=TRUE)
```



Sample-Wise Clustering with `rlog` Values

```
> library(DESeq2)
> countDF <- as.matrix(read.table("./results/countDFeByg.xls"))
> colData <- data.frame(row.names=targets$SampleName, condition=targets$Factor)
> dds <- DESeqDataSetFromMatrix(countData = countDF, colData = colData, design = ~ condition)
> d <- cor(assay(rlog(dds)), method="spearman")
> hc <- hclust(dist(1-d))
> plot.phylo(as.phylo(hc), type="p", edge.col=4, edge.width=3, show.node.label=TRUE, no.margin=TRUE)
```



Exercise 1: *QuasR* with Antisense Read Counting

Task 1 Align reads from all 4 samples.

Task 2 Count reads in sense and antisense. Discuss differences. Why is this analysis meaningless for the provided non-strand-specific RNA-Seq samples?

Task 3 Identify all genes where the antisense counts are $\geq$3-fold higher than the sense counts in at least 2 out of the 4 samples.

Task 4 Plot the result of the most pronounced antisense expression case with *ggbio*.

Outline

Identify DEGs with Simple Fold Change Method

Compute mean values for replicates

```
> source("http://faculty.ucr.edu/~tgirke/Documents/R_BioCond/My_R_Scripts/colAg.R")
> M12_A12 <- c("M12A", "M12B", "A12A", "A12B")
> rpkmDFeByg <- read.table("./results/rpkmDFeByg.xls")[, M12_A12]
> countDFrpkm_mean <- colAg(myMA=rpkmDFeByg, group=c(1,1,2,2), myfct=mean)
> countDFrpkm_mean[1:4,]
```

```
          M12A_M12B A12A_A12B
AT1G01010 3557.1959 4279.9292
AT1G01020 1520.5605 1054.1279
AT1G01030  766.2811  495.6289
AT1G01040 3488.4589 2245.1415
```

Log2 fold changes

```
> countDFrpkm_mean <- cbind(countDFrpkm_mean, log2ratio=log2(countDFrpkm_mean[,1]/countDFrpkm_mean[,2]))
> countDFrpkm_mean <- countDFrpkm_mean[is.finite(countDFrpkm_mean[,3]), ]
> degs2fold <- countDFrpkm_mean[countDFrpkm_mean[,3] >= 1 | countDFrpkm_mean[,3] <= -1,]
> degs2fold[1:4,]
```

```
           M12A_M12B A12A_A12B log2ratio
AT1G01050 25903.2856 7901.1976  1.712992
AT1G01070   723.8589  138.3461  2.387427
AT1G01080 15247.6545 2608.0398  2.547550
AT1G01090 35151.8743 8586.1872  2.033512
```

```
> write.table(degs2fold, "./results/degs2fold.xls", quote=FALSE, sep="\t", col.names = NA)
> degs2fold <- read.table("./results/degs2fold.xls")
```

Identify DEGs with *DESeq2* Library

Raw count data are expected here!

```
> library(DESeq2)
> countDF <- read.table("./results/countDFeByg.xls")[, M12_A12]
> countData <- as.matrix(countDF)
> colData <- data.frame(condition=c(M12A="M12", M12B="M12", A12A="A12", A12B="A12"))
> dds <- DESeqDataSetFromMatrix(countData = countData, # Create DESeqDataSet object
+                               colData = colData, design = ~ condition)
> dds <- DESeq(dds) # (i) estimation of size factors, (ii) dispersion,
>                   # (iii) negative binomial GLM fitting and (iv) Wald statistics
> res <- results(dds) # Extracts DEG results from DESeqDataSet object
> res[is.na(res[,"padj"]), "padj"] <- 1
> res <- na.omit(res)
> res2fold <- res[res$log2FoldChange >= 1 | res$log2FoldChange <= -1,]
> res2foldpadj <- res2fold[res2fold$padj <= 0.2, ]
> res2foldpadj[1:4,]
```

```
log2 fold change (MAP): condition M12 vs A12
Wald test p-value: condition M12 vs A12
DataFrame with 4 rows and 6 columns
            baseMean log2FoldChange     lfcSE      stat      pvalue       padj
           <numeric>      <numeric> <numeric> <numeric>   <numeric>  <numeric>
AT1G01080   79.53825       1.385699 0.5954125  2.327293 0.019949670 0.07979868
AT1G01090  252.22561       1.011465 0.5285502  1.913660 0.055663666 0.15289865
AT2G01021 3466.65548      -1.373892 0.5714111 -2.404384 0.016199752 0.07979868
ATCG00020  972.84314      -1.899092 0.6606360 -2.874642 0.004044864 0.02696576
```

Identify DEGs with *edgeR's* Exact Method

DEG analysis with classical *edgeR* approach. Note: raw read count data are expected by all methods!

```
> library(edgeR)
> countDF <- read.table("./results/countDFeByg.xls")[, M12_A12]
> conds <- colData$condition
> y <- DGEList(counts=countDF, group=conds) # Constructs DGEList object
> y <- estimateCommonDisp(y) # Estimates common dispersion
> y <- estimateTagwiseDisp(y) # Estimates tagwise dispersion
> et <- exactTest(y, pair=c("M12", "A12")) # Computes exact test for the negative binomial distribution.
> topTags(et, n=4)

Comparison of groups:  A12-M12
             logFC   logCPM       PValue       FDR
AT1G01080 -2.532293 13.56691 0.0008697014 0.1008854
AT3G01060 -2.088054 12.06507 0.0023504132 0.1363240
AT1G01090 -2.032593 15.13123 0.0064141533 0.2480139
AT5G01020 -1.747026 14.73618 0.0124241193 0.3121957

> edge <- as.data.frame(topTags(et, n=50000))
> edge2fold <- edge[edge$logFC >= 1 | edge$logFC <= -1,]
> edge2foldpadj <- edge2fold[edge2fold$FDR <= 0.2, ]
```

Identify DEGs with *edgeR's* GLM Approach

DEG analysis with *edgeR* using generalized linear models (glms)

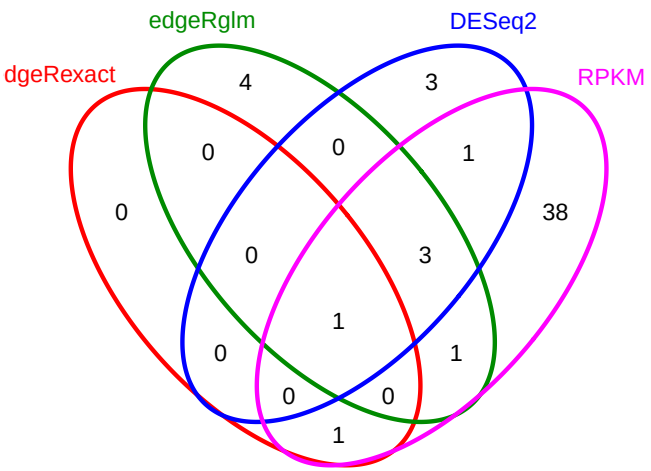
```
> library(edgeR)
> countDF <- read.table("./results/countDFeByg.xls")[, M12_A12]
> y <- DGEList(counts=countDF, group=conds) # Constructs DGEList object
> ## Filtering and normalization
> keep <- rowSums(cpm(y)>1) >= 2; y <- y[keep, ]
> y <- calcNormFactors(y)
> design <- model.matrix(~0+group, data=y$samples); colnames(design) <- levels(y$samples$group) # Design matrix
> ## Estimate dispersion
> y <- estimateGLMCommonDisp(y, design, verbose=TRUE) # Estimates common dispersions

Disp = 0.1936 , BCV = 0.44

> y <- estimateGLMTrendedDisp(y, design) # Estimates trended dispersions
> y <- estimateGLMTagwiseDisp(y, design) # Estimates tagwise dispersions
> ## Fit the negative binomial GLM for each tag
> fit <- glmFit(y, design) # Returns an object of class DGEGLM
> contrasts <- makeContrasts(contrasts="M12-A12", levels=design) # Contrast matrix is optional
> lrt <- glmLRT(fit, contrast=contrasts[,1]) # Takes DGEGLM object and carries out the likelihood ratio test.
> edgeglm <- as.data.frame(topTags(lrt, n=length(rownames(y))))
> ## Filter on fold change and FDR
> edgeglm2fold <- edgeglm[edgeglm$logFC >= 1 | edgeglm$logFC <= -1,]
> edgeglm2foldpadj <- edgeglm2fold[edgeglm2fold$FDR <= 0.2, ]
```

Comparison Among DEG Results

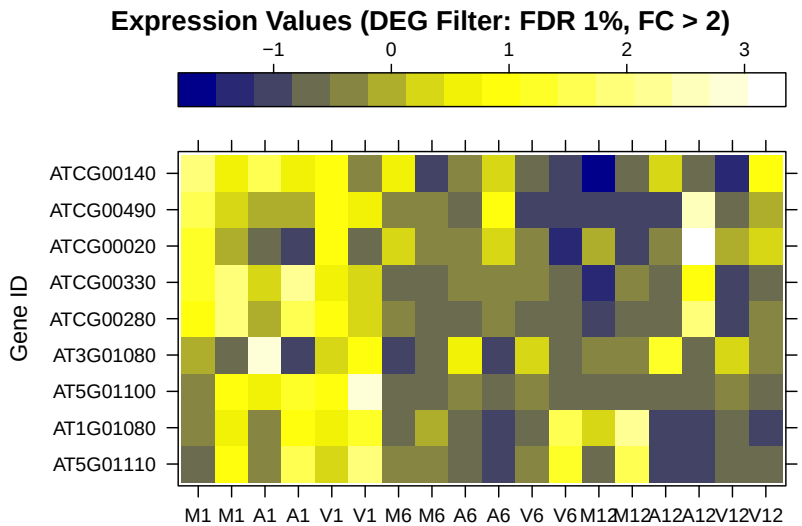
```
> source("http://faculty.ucr.edu/~tgirke/Documents/R_BioCond/My_R_Scripts/overLapper.R")
> setlist <- list(edgeRexact=rownames(edge2foldpadj), edgeRglm=rownames(edgeglm2foldpadj), DESeq2=rownames(res2
> OLlist <- overLapper(setlist=setlist, sep="_", type="vennsets")
> counts <- sapply(OLlist$Venn_List, length)
> vennPlot(counts=counts, mymain="DEG Comparison")
```



Heatmap of Top Ranking DEGs

Note: gene-wise clustering is not possible with a single sample pair. The following shows the scaled expression values (here RPKMs) in form of a heatmap.

```
> library(lattice); library(gplots)
> rpkmDFeByg <- read.table("./results/rpkmDFeByg.xls")
> y <- rpkmDFeByg[rownames(edgeglm2foldpadj),]
> colnames(y) <- targets$Factor
> y <- t(scale(t(as.matrix(y))))
> y <- y[order(y[,1]),]
> levelplot(t(y), height=0.2, col.regions=colorpanel(40, "darkblue", "yellow", "white"), main="Expression Value
```

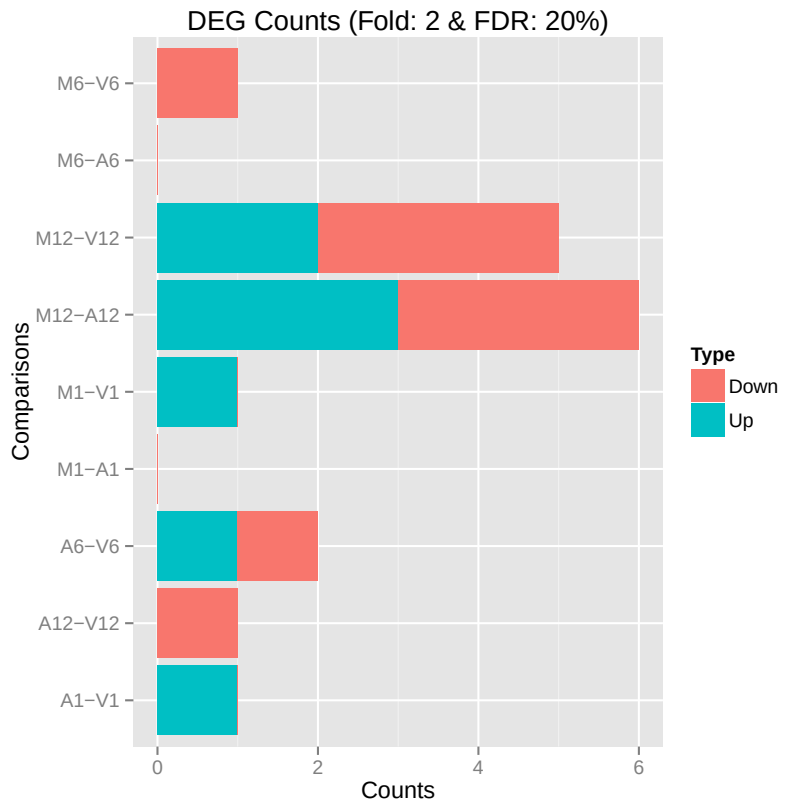


DEGs for all Comparisons Defined in `targets` File

```
> countDF <- read.table("./results/countDFeByg.xls")
> cmp <- readComp(file="targets.txt", format="matrix", delim="-")
> edgeDF <- run_edgeR(countDF=countDF, targets=targets, cmp=cmp[[1]], independent=FALSE, mdsplot="")
```

```
Disp = 0.17287 , BCV = 0.4158
```

```
> DEG_list <- filterDEGs(degDF=edgeDF, filter=c(Fold=2, FDR=20))
```



Outline

Enrichment of GO Terms in DEG Sets

The following performs GO term enrichment analysis of one of the identified DEG sets using the *GOstats* Link package. Another package, among many others, to consider here is the *goseq* Link that considers gene length bias in RNA-Seq data.

```
> library(GOstats); library(GO.db); library(ath1121501.db)
> geneUniverse <- rownames(countDF)
> geneSample <- rownames(res2foldpadj)
> params <- new("GOHyperGParams", geneIds = geneSample, universeGeneIds = geneUniverse,
+         annotation="ath1121501", ontology = "MF", pvalueCutoff = 0.5,
+         conditional = FALSE, testDirection = "over")
> hgOver <- hyperGTest(params)
> summary(hgOver)[c(1,2,4,5),]
```

	GOMFID	Pvalue	OddsRatio	ExpCount	Count	Size	Term
1	GO:0016168	0.0001127453	94.66667	0.4430380	4	5	chlorophyll binding
2	GO:0046906	0.0003273065	46.66667	0.5316456	4	6	tetrapyrrole binding
4	GO:0097159	0.0589172699	5.00000	2.5696203	5	29	organic cyclic compound binding
5	GO:1901363	0.0589172699	5.00000	2.5696203	5	29	heterocyclic compound binding

```
> htmlReport(hgOver, file = "results/MyhyperGresult.html")
```

Batch GO Term Enrichment Analysis (Part I)

The following shows how to obtain gene-to-GO mappings from *biomaRt*. This is relatively slow, but it needs to be done only once.

```
> 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=NU
> save(catdb, file="data/GO/catdb.RData")
```

Batch GO Term Enrichment Analysis (Part II)

The 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=0.
> 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=gosl
```

Batch GO Term Enrichment Analysis (Part III)

Plot batch GO term results

```
> goBarplot(BatchResultslim, gocat="MF")
```

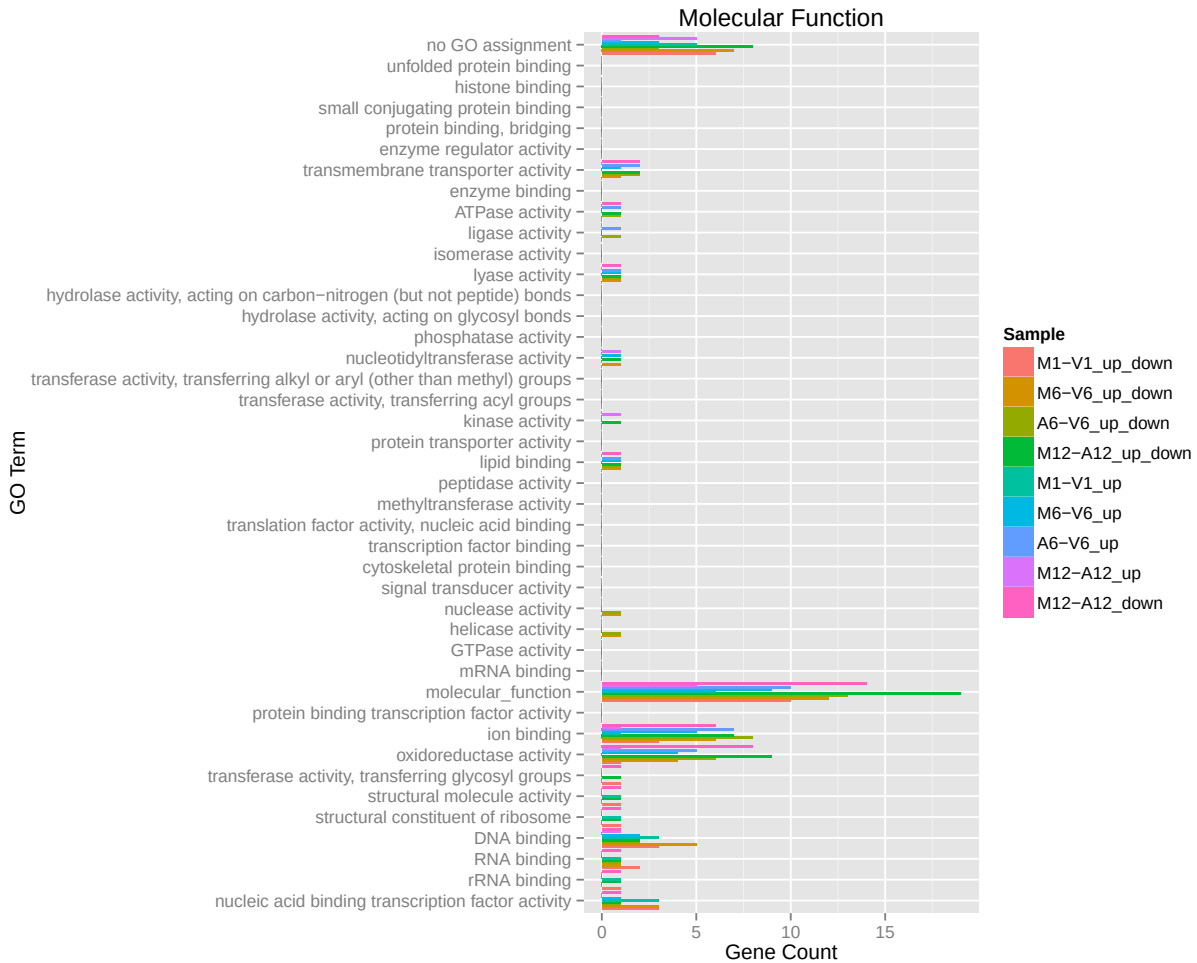


Figure: GO Slim Barplot for MF Ontology.

Outline

Inspect Results in IGV

View results in IGV

- Download and open IGV Link
- Select in menu in top left corner `A. thaliana (TAIR10)`
- Upload the following indexed/sorted Bam files with `File -> Load from File...`

```
./data/SRR446039_1.fastq.tophat/accepted_hits.bam
./data/SRR446040_1.fastq.tophat/accepted_hits.bam
./data/SRR446041_1.fastq.tophat/accepted_hits.bam
./data/SRR446042_1.fastq.tophat/accepted_hits.bam
```

- To view area of interest, enter its coordinates `Chr1:45,296-47,019` in position menu on top.

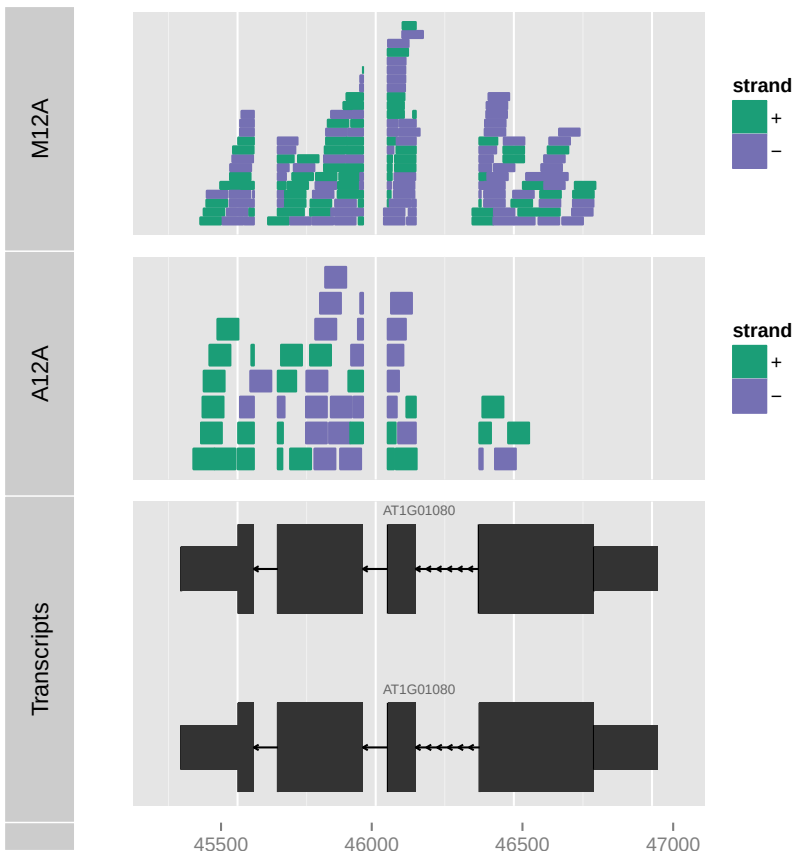
Controlling IGV from R

Create previous IGV session with required tracks automatically, and direct it to a specific position, here `Chr1:45,296-47,019`.

```
> library(SRAdb)
> startIGV("lm")
> sock <- IGVsocket()
> session <- IGVsession(files=outpaths(args)[M12_A12],
+                       sessionFile="session.xml",
+                       genome="A. thaliana (TAIR10)")
> IGVload(sock, session)
> IGVgoto(sock, 'Chr1:45296-47019')
```

Generate Similar View with *ggbio* Programmatically

```
> library(ggbio)
> M12A <- readGAlignmentsFromBam(outpaths(args)["M12A"], use.names=TRUE, param=ScanBamParam(which=GRanges("Chr1
> A12A <- readGAlignmentsFromBam(outpaths(args)["A12A"], use.names=TRUE, param=ScanBamParam(which=GRanges("Chr1
> p1 <- autoplot(M12A, geom = "rect", aes(color = strand, fill = strand))
> p2 <- autoplot(A12A, geom = "rect", aes(color = strand, fill = strand))
> p3 <- autoplot(txdb, which=GRanges("Chr1", IRanges(45296, 47019)), names.expr = "gene_id")
> tracks(M12A=p1, A12A=p2, Transcripts=p3, heights = c(0.3, 0.3, 0.4)) + ylab("")
```



Exercise 2: Venn Diagram for Up/Down DEGs

Task 1 Store the identifiers of the upregulated genes from each of the four DEG methods in separate components of a list. Note: the definition of up and down is arbitrary and one needs to check how it is defined by the different DEG methods!

Task 2 Do the same for the downregulated genes.

Task 3 Compare the overlaps among the different up/down sets in a single 4-way venn diagram.

Outline

Analysis of Differential Exon Usage with DEXSeq

Number of reads overlapping gene ranges

```
> library(DEXSeq)
> exonicParts <- disjointExons(txdb, aggregateGenes=FALSE )
> bamlst <- BamFileList(outpaths(args)[M12_A12], index=character(), yieldSize=100000, obeyQname=TRUE)
> SE <- summarizeOverlaps(exonicParts, bamlst, mode="Union", singleEnd=TRUE, ignore.strand=TRUE, inter.feature=
> colData(SE)$condition <- colData$condition
> dxd <- DEXSeqDataSetFromSE(SE, design= ~ sample + exon + condition:exon)
> featureCounts(dxd)[1:2,] # Counts for individual exons

               M12A M12B A12A A12B
AT1G01010:E001    9   12   14    6
AT1G01010:E002   12   15   12   11

> assays(dxd)$counts[1:2,] # Counts for individual exons plus for all remaining exons of a gene

               [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
AT1G01010:E001    9   12   14    6   59   71   67   37
AT1G01010:E002   12   15   12   11   56   68   69   32

> # rowData(dxd)[1:4,] # Exon ranges
> # colData(dxd) # Sample data
> write.table(featureCounts(dxd), "./results/countDFdex", quote=FALSE, sep="\t", col.names = NA)
```

Analysis of Differential Exon Usage with DEXSeq

Identify genes with differential exon usage

```
> ## Performs normalization
> dxd <- estimateSizeFactors(dxd)
> ## Evaluate variance of the data by estimating dispersion using Cox-Reid (CR) likelihood estimation
> dxd <- estimateDispersions(dxd)
> ## Performs Chi-squared test on each exon and Benjmini-Hochberg p-value adjustment for mutliple testing
> dxd <- testForDEU(dxd)
> ## Estimates fold changes of exons
> dxd <- estimateExonFoldChanges(dxd, fitExpToVar="condition")
> ## Obtain results as DataFrame
> dxr1 <- DEXSeqResults(dxd)
> ## Column descriptions
> col_descr <- elementMetadata(dxr1)$description
> ## Count number of genes with differential exon usage
> dxr1[is.na(dxr1$padj), "padj"] <- 1
> table(tapply(dxr1$padj < 0.2, dxr1$groupID, any))

FALSE  TRUE
  115     1

> ## DEU sample
> dxr1[dxr1$groupID=="AT4G00050",][1:4, c(1:2,7:10)]

DataFrame with 4 rows and 6 columns
                   groupID   featureID       padj       A12       M12 log2fold_A12_M12
                 <character> <character>  <numeric> <numeric> <numeric>        <numeric>
AT4G00050:E001   AT4G00050        E001 1.00000000  6.275199  6.688707      -0.09206611
AT4G00050:E002   AT4G00050        E002 0.08536468 12.137741 10.860877       0.16035925
AT4G00050:E003   AT4G00050        E003 1.00000000  4.998105  5.463383      -0.12841322
AT4G00050:E004   AT4G00050        E004 1.00000000  4.793227  5.288418      -0.14183906
```

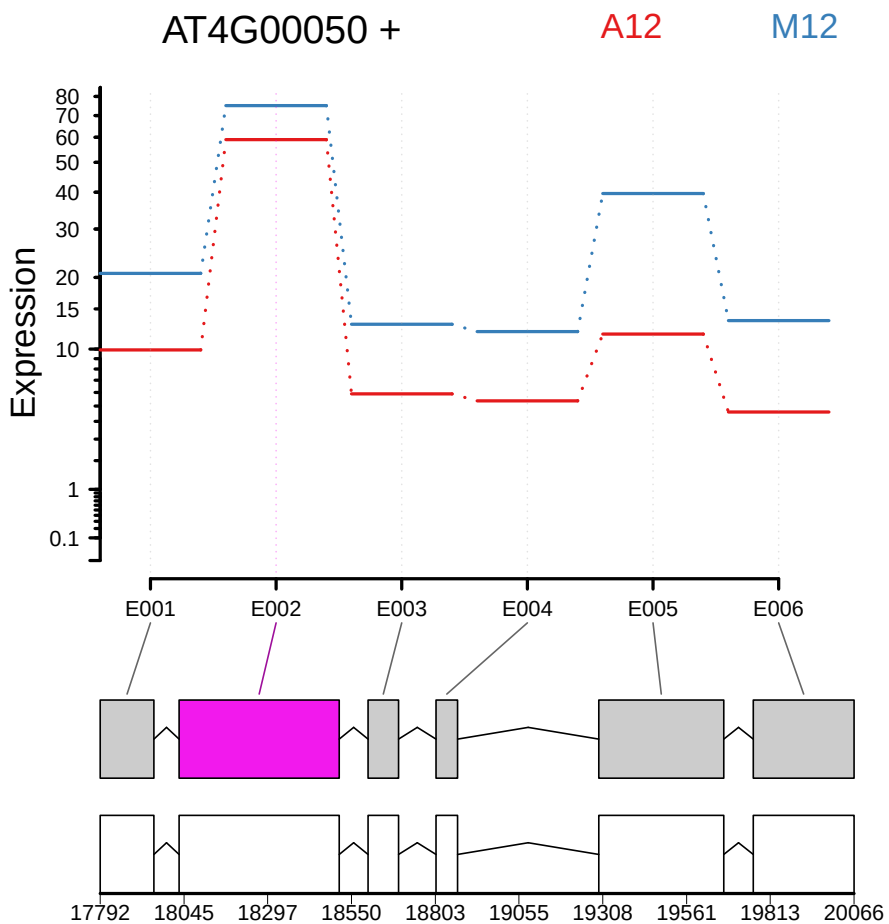
DEXSeq Plots

Sample plot showing fitted expression of exons

```
> plotDEXSeq(dxr1, "AT4G00050", displayTranscripts=TRUE, legend=TRUE, cex.axis=1.2, cex=1.3, lwd=2 )
```

Generate many plots and write them to results directory

```
> mygeneIDs <- unique(as.character(na.omit(dxr1[dxr1$groupID %in% unique(dxr1$groupID),])[,"groupID"]))
> DEXSeqHTML(dxr1, genes=mygeneIDs[1:10], path="results", file="DEU.html")
```



systemPipeR: Run Entire RNA-Seq Workflow and Generate Report

- *systemPipeR* is useful for building end-to-end analysis pipelines with automated report generation for NGS applications such as RNA-Seq and many others.
- It provides 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.
- To generate the report for the data sets and analysis steps demonstrated in this tutorial, open the file `systemPipeR.Rnw` Link in RStudio's code editor and then click the Compile PDF button. This will run the entire analysis and generate the corresponding RNA-Seq analysis report (PDF format) along with a bibliography of citations included in the text.
- Alternatively, one can achieve the same result by running the following commands from the command-line:

```
echo 'Sweave("systemPipeR.Rnw")' | R --slave # Runs R code
echo 'Stangle("systemPipeR.Rnw")' | R --slave # Extracts R code
pdflatex systemPipeR.tex; bibtex systemPipeR; pdflatex systemPipeR.tex # Compiles PDF
```

- Note: for time reasons, not all code chunks are evaluated (change `eval=FALSE` to `eval=TRUE`) when the report is generated.
- A sample report can be viewed here systemPipeR.pdf Link
- To efficiently customize these reports, users want to learn how to use *Latex/Sweave* and/or *knitr*.

Session Information

```
> sessionInfo()

R version 3.1.2 (2014-10-31)
Platform: x86_64-unknown-linux-gnu (64-bit)
locale:
[1] C

attached base packages:
[1] parallel  stats4    stats     graphics  utils     datasets  grDevices methods   base

other attached packages:
 [1] DEXSeq_1.12.1           ggbio_1.14.0            ggplot2_1.0.0           xtable_1.7-4            ath112150
[10] GO.db_3.0.0             Matrix_1.1-4            gplots_2.14.2           lattice_0.20-29         edgeR_3.8
[19] ape_3.1-4               GenomicFeatures_1.18.2  rtracklayer_1.26.2      systemPipeR_1.0.0       Annotatio
[28] GenomicAlignments_1.2.1 BiocParallel_1.0.0      Rsamtools_1.18.2        Biostrings_2.34.0       XVector_0
[37] BiocGenerics_0.12.1

loaded via a namespace (and not attached):
 [1] AnnotationForge_1.8.1    BBmisc_1.8               BSgenome_1.34.0          BatchJobs_1.5            Formu
 [9] KernSmooth_2.23-13       MASS_7.3-35              OrganismDbi_1.8.0        RBGL_1.42.0              RColo
[17] acepack_1.3-3.3          annotate_1.44.0          base64enc_0.1-2          biomaRt_2.22.0           biovi
[25] checkmate_1.5.0          cluster_1.15.3           codetools_0.2-9          colorspace_1.2-4         dichr
[33] foreign_0.8-61           gdata_2.13.3             genefilter_1.48.1        geneplotter_1.44.0       grid_
[41] hwriter_1.3.2            iterators_1.0.7          labeling_0.3             latticeExtra_0.6-26      locfi
[49] pheatmap_0.7.7           plyr_1.8.1               proto_0.3-10             reshape_0.8.5            resha
[57] sendmailR_1.2-1          splines_3.1.2            statmod_1.4.20           stringr_0.6.2            survi
```

Outline

References I

Anders, S., Huber, W., 2010. Differential expression analysis for sequence count data. Genome Biol 11 (10).
URL http://www.hubmed.org/display.cgi?uids=20979621

Anders, S., Reyes, A., Huber, W., Oct 2012. Detecting differential usage of exons from RNA-seq data. Genome Res 22 (10), 2008–2017.
URL http://www.hubmed.org/display.cgi?uids=22722343

Howard, B. E., Hu, Q., Babaoglu, A. C., Chandra, M., Borghi, M., Tan, X., He, L., Winter-Sederoff, H., Gassmann, W., Veronese, P., Heber, S., 1 Oct. 2013. High-throughput RNA sequencing of pseudomonas-infected arabidopsis reveals hidden transcriptome complexity and novel splice variants. PLoS One 8 (10), e74183.
URL http://dx.doi.org/10.1371/journal.pone.0074183

Robinson, M. D., McCarthy, D. J., Smyth, G. K., Jan 2010. edgeR: a Bioconductor package for differential expression analysis of digital gene expression data. Bioinformatics 26 (1), 139–140.
URL http://www.hubmed.org/display.cgi?uids=19910308

Robinson, M. D., Oshlack, A., Mar 2010. A scaling normalization method for differential expression analysis of RNA-seq data. Genome Biol 11 (3).
URL http://www.hubmed.org/display.cgi?uids=20196867

Trapnell, C., Hendrickson, D. G., Sauvageau, M., Goff, L., Rinn, J. L., Pachter, L., Jan 2013. Differential analysis of gene regulation at transcript resolution with RNA-seq. Nat Biotechnol 31 (1), 46–53.
URL http://www.hubmed.org/display.cgi?uids=23222703