Introduction to differential gene expression analysis using RNA-seq

Written by Friederike Dündar, Luce Skrabanek, Paul Zumbo

September 2015
updated October 8, 2018

1 Introduction to RNA-seq **5**
1.1 RNA extraction 5
1.1.1 Quality control of RNA preparation (RIN) 6
1.2 Library preparation methods 6
1.3 Sequencing (Illumina) 9
1.4 Experimental Design 10
1.4.1 Capturing enough variability 11
1.4.2 Avoiding bias 12

2 Raw Data (Sequencing Reads) **14**
2.1 Download from ENA 14
2.2 Storing sequencing reads: `FASTQ` format 15
2.3 Quality control of raw sequencing data 18

3 Read Alignment **21**
3.1 Reference genomes and annotation 21
3.1.1 File formats for defining genomic regions 22
3.2 Aligning reads using `STAR` 26
3.3 Storing aligned reads: `SAM/BAM` file format 27
3.3.1 The `SAM` file header section 28
3.3.2 The `SAM` file alignment section 30
3.3.3 Manipulating `SAM/BAM` files 34
3.4 Quality control of aligned reads 36
3.4.1 Basic alignment assessments 36
3.4.2 Bias identification 41
3.4.3 Quality control with QoRTs 44
3.4.4 Summarizing the results of different QC tools with `MultiQC` 44

4 Read Quantification **46**
4.1 Gene-based read counting 46
4.2 Isoform counting methods 47

5 Normalizing and Transforming Read Counts **50**
5.1 Normalization for sequencing depth differences 50
5.1.1 `DESeq`'s specialized data set object 50
5.2 Transformation of sequencing-depth-normalized read counts 53
5.2.1 Log_2 transformation of read counts 53
5.2.2 Visually exploring normalized read counts 53
5.2.3 Transformation of read counts including variance shrinkage 54
5.3 Exploring global read count patterns 55
5.3.1 Pairwise correlation 55
5.3.2 Hierarchical clustering 55
5.3.3 Principal Components Analysis (PCA) 57

6 Differential Gene Expression Analysis **59**
6.1 Estimating the difference between read counts for a given gene . . . 60
6.2 Testing the null hypothesis . . . 61
6.3 Running DGE analysis tools . . . 61
6.3.1 DESeq2 workflow . . . 61
6.3.2 Exploratory plots following DGE analysis . . . 62
6.3.3 Exercise suggestions . . . 65
6.3.4 edgeR . . . 65
6.3.5 `limma-voom` . . . 66
6.4 Judging DGE results . . . 68
6.5 Example downstream analyses . . . 69

7 Appendix **71**
7.1 Improved alignment . . . 71
7.2 Additional tables . . . 72
7.3 Installing bioinformatics tools on a UNIX server . . . 82

© 2015-2018 Applied Bioinformatics Core | Weill Cornell Medical College

List of Tables

1 Sequencing depth recommendations. 11
2 Examples for technical and biological replicates. 11
3 Illumina's different base call quality score schemes. 17
4 The fields of the alignment section of `SAM` files. 31
5 The `FLAG` field of `SAM` files. 32
6 Comparison of classification and clustering techniques. 56
7 Programs for DGE. 59
8 Biases and artifacts of Illumina sequencing data. 72
9 `FASTQC` test modules. 73
10 Optional entries in the header section of `SAM` files. 75
11 Overview of `RSeQC` scripts. 76
12 Overview of `QoRTs` QC functions. 78
13 Normalizing read counts between different conditions. 80
14 Normalizing read counts within the same sample. 81

List of Figures

1 RNA integrity assessment (RIN). 6
2 Overview of general RNA library preparation steps. 7
3 Size selection steps during common RNA library preparations. 8
4 The different steps of sequencing by synthesis. 9
5 Sequence data repositories. 14
6 Phred score ranges. 17
7 Typical bioinformatics workflow of differential gene expression analysis. 18
8 RNA-seq read alignment. 21
9 Schematic representation of a `SAM` file. 28
10 CIGAR strings of aligned reads. 34
11 Graphical summary of `STAR`'s log files for 96 samples. 38
12 Different modes of counting read-transcript overlaps. 46
13 Schema of a simple *deBruijn* graph-based transcrip determination. 48
14 Effects of different read count normalization methods. 52
15 Comparison of the read distribution plots for untransformed and log_2-transformed values. . . 53
16 Comparison of log_2- and *rlog*-transformed read counts. 54
17 Dendrogram of *rlog*-transformed read counts. 57
18 PCA on raw counts and *rlog*-transformed read counts. 58
19 Histogram and MA plot after DGE analysis. 62
20 Heatmaps of *log*-transformed read counts. 63
21 Read counts for two genes in two conditions. 64
22 Example plots to judge DGE analysis results. 70

Technical Prerequisites

Command-line interface The first steps of the analyses – that are the most computationally demanding – will be performed directly on our servers. The interaction with our servers is completely text-based, i.e., there will be no graphical user interface. We will instead be communicating entirely via the command line using the UNIX shell scripting language `bash`. You can find a good introduction into the `shell` basics at `http://linuxcommand.org/lc3_learning_the_shell.php` (for our course, chapters 2, 3, 5, 7, and 8 are probably most relevant).

To start using the command line, Mac users should use the App called `Terminal`. Windows users need to install `putty`, a Terminal emulator (`http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html`). You probably want the bits under the *A Windows installer for everything except PuTTYtel* heading. Putty will allow you to establish a connection with a UNIX server and interact with it.

Programs that we will be using via the command line:

`FastQC`	`http://www.bioinformatics.babraham.ac.uk/projects/fastqc/`
`featureCounts`	`http://bioinf.wehi.edu.au/subread-package/)`
`MultiQC`	`http://multiqc.info/docs/`
`QoRTs`	`https://hartleys.github.io/QoRTs/`
`RSeQC`	`http://rseqc.sourceforge.net/`
`samtools`	`http://www.htslib.org/`
`STAR`	`https://github.com/alexdobin/STAR`
`UCSC tools`	`https://hgdownload.soe.ucsc.edu/admin/exe/`

Details on how to install these programs via the command line can be found in the Appendix.

The only program with a graphical user interface will be `IGV`. Go to `https://www.broadinstitute.org/igv/` → "Downloads", register with your academic email address and launch the Java web start (for Windows machines, you should go for the 1.2 GB version).

R The second part of the analyses – where we will need support for statistics and visualization more than pure computation power – will mostly be done on the individual computers using the programming language R. You can download R for both MacOS and Windows from `http://cran.rstudio.com/`. After you have installed R, we highly recommend to install RStudio (`http://www.rstudio.com/products/rstudio/download/`), which will provide you with an interface to write commands at a prompt, construct a script and view plots all in a single integrated environment.

R packages that will be used throughout the course:

from CRAN:	`ggplot2, magrittr, pheatmap, UpSetR`
from bioconductor:	`DESeq2, edgeR, ggplot2, limma, pcaExplorer, org.Sc.scd.db, vsn`

© 2015-2018 Applied Bioinformatics Core | Weill Cornell Medical College

1 Introduction to RNA-seq

The original goal of RNA sequencing was to identify which genomic loci are expressed in a cell (population) at a given time over the entire expression range, i.e., to offer a superior alternative to cDNA microarrays. Indeed, RNA-seq was shown to detect lowly expressed transcripts while suffering from strongly reduced false positive rates in comparison to microarray based expression quantification (Illumina, 2011; Nookaew et al., 2012; Zhao et al., 2014)*. Since RNA-seq does not rely on a pre-specified selection of cDNA probes, there are numerous additional applications of RNA-seq that go beyond the counting of expressed transcripts of known genes, such as the detection and quantification of non-genic transcripts, splice isoforms, novel transcripts and protein-RNA interaction sites. The detection of gene expression changes (i.e., mRNA levels) between different cell populations and/or experimental conditions remains the most common application of RNA-seq. Nevertheless, unlike for genome-based high-throughput sequencing approaches, the RNA-seq field lacks widely accepted and adopted standards and is only slowly coming to terms about best practices for differential gene expression analysis amid a myriad of available software (Byron et al., 2016).

The general workflow of a differential gene expression analysis is:

1. **Sequencing (biochemistry)**
   (a) RNA extraction
   (b) Library preparation (including mRNA enrichment)
   (c) Sequencing
2. **Bioinformatics**
   (a) Processing of sequencing reads (including alignment)
   (b) Estimation of individual gene expression levels
   (c) Normalization
   (d) Identification of differentially expressed (DE) genes

1.1 RNA extraction

Before RNA can be sequenced, it must first be extracted and separated from its cellular environment, which consists primarily of proteins and DNA. The most prevalent methods for RNA isolation are silica-gel based membranes or liquid-liquid extractions with *acidic* phenol-chloroform. In the former case, RNA exclusively binds to a silica-gel membrane, while the remaining cellular components are washed away. Silica-gel membranes require ethanol for binding. The volume of ethanol influences which transcripts are bound to the membrane: more ethanol results in the retention of RNAs <200 bp, whereas a smaller volume results in their loss. When using phenol-chloroform extraction, the cellular components are dissolved into three phases: the organic phase; the interphase; and the aqueous phase, in which the RNA is retained. Phenol-chloroform extraction is typically followed by an alcohol precipitation to de-salt and concentrate the RNA. An alcohol precipitation can be performed with either ethanol or isopropanol, both of which require the use of a salt. Different salts lead to different precipitation efficiencies and result in different RNA populations; e.g., lithium chloride, a commonly used salt, has been reported to result in the loss of tRNAs, 5S rRNAs, snRNAs, and other RNAs <250–300 bp (Cathala et al., 1983). Given the multitude of factors that can influence the outcome of RNA extraction, it is therefore important to process the RNA in a highly controlled and standardized manner, so that the knowledge of how the RNA was isolated can be appropriately leveraged for one's understanding of the data later on. Additional information on how RNA extraction methods influence RNA-seq data can be found in Sultan et al. (2014).

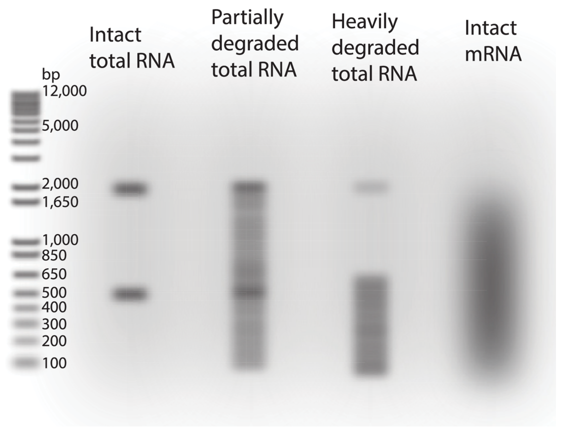
Although both extraction methods previously mentioned are designed to eliminate DNA contamination, they are often imperfect. But even small amounts of DNA contamination (as little as 0.01% genomic DNA by weight) can negatively impact results (NuGEN, 2013). Accordingly, it is advisable to take additional measures to ensure DNA-free RNA, e.g., by treating the RNA with DNase.

*For a detailed comparison of different methods for transcriptome quantification, see Lowe et al. (2017)

1.1.1 Quality control of RNA preparation (RIN)

RNA is much more susceptible to degradation than DNA and the quality of the extracted RNA molecules can strongly impact the results of the RNA-seq experiment. Traditionally, RNA integrity was assessed via gel electrophoresis by visual inspection of the ribosomal RNA bands. Intact eukaryotic total RNA should yield clear 28S and 18S rRNA bands. The 28S rRNA band is approximately twice as intense as the 18S rRNA band (2:1 ratio). As RNA degrades, the 2:1 ratio of high quality RNA decreases, and low molecular weight RNA begins to accumulate (Figure 1a). Since the human interpretation of gel images is subjective and has been shown to be inconsistent, Agilent developed a software algorithm that allows for the calculation of an RNA Integrity Number (RIN) from a digital representation of the size distribution of RNA molecules (which can be obtained from an Agilent Bioanalzyer). The RIN number is based on a numbering system from 1 to 10, with 1 being the most degraded and 10 being the most intact (Figure 1b). This approach facilitates the interpretation and reproducibility, of RNA quality assessments, and provides a means by which samples can be compared in a standardized manner.

(a) Gel electropherogram



(b) Capillary electropherogram

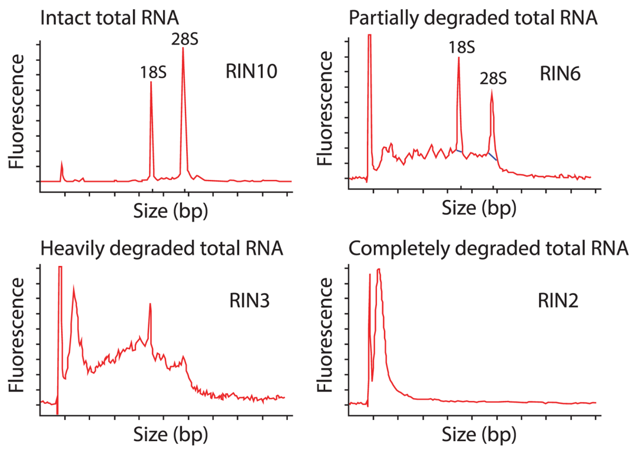


Figure 1: RNA integrity assessment is based on the ratio of $\frac{28S}{18S}$ rRNA, estimated from the band intensity (a) or a densitometry plot (b). RNA used for RNA-seq experiments should be as intact as possible. Figure taken from Griffith et al. (2015).

1.2 Library preparation methods

In high-throughput sequencing terms, a *library* is a (preferably random) collection of DNA *fragments* that are ready for sequencing with a specific protocol (Figure 2). For Illumina-based protocols, cDNA fragments will typically be between 150 to 300 bp, and after hybridization to the *flowcell*, the ends (50 to 150 bp) of these fragments will be sequenced. For a comprehensive overview of recent high-throughput sequencing methods *beyond Illumina protocols*, see Goodwin et al. (2016).

Due to the numerous types of RNA families, there is a great variety of library preparation protocols. Since the quantification of mRNA is by far the most commonly used application of RNA-seq experiments, we will focus on protocols that are typically applied in this context. Keep in mind that the library preparation can seriously affect the outcome of the sequencing in terms of quality as well as coverage. More importantly, small transcripts (smaller than about 150 bp) and strand information will be lost during standard RNA-seq library preparations (Figure 3), i.e., if those details are of interest to you, make sure to select an alternative protocol. For more details on library preparation protocols including single-cell RNA-seq, CLiP-seq and more, see Head et al. (2014), Shanker et al. (2015), and Yeri et al. (2018).

© 2015-2018 Applied Bioinformatics Core | Weill Cornell Medical College

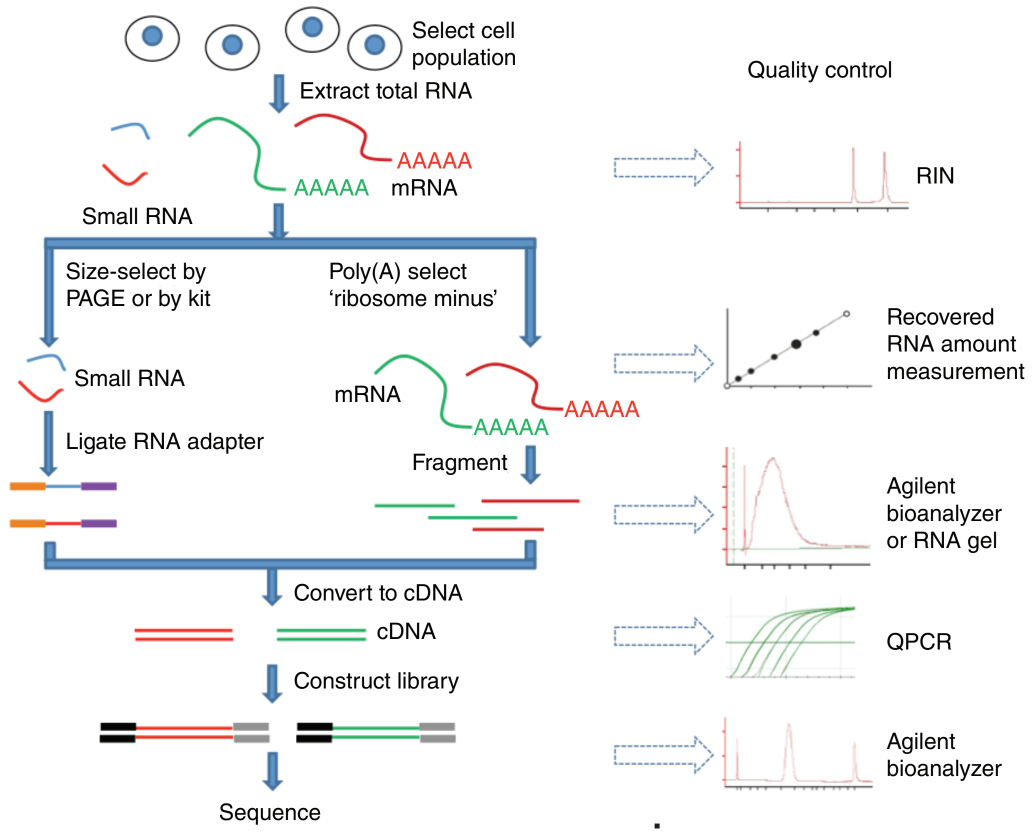


Figure 2: General RNA library preparation workflow. After RNA extraction and measuring its integrity, rRNA is depleted (either using poly(A)-selection or rRNA depletion) and the remaining RNA molecules are fragmented, ideally achieving a uniform size distribution. Double-stranded cDNA is synthesized and the adapters for sequencing are added to construct the final library whose fragment size distribution should be unimodal and well-defined. Image taken from Zeng and Mortazavi (2012).

mRNA enrichment Since extracted RNA contains 80–85% rRNA and 10–15% tRNA (Farrell, 2010), mRNA needs to be enriched for. This is typically done either via enrichment of poly(A)-tail containing nucleic acids using oligo(dT) beads or by removal of ribosomal RNA via complementary sequences. This means that various non-polyadenylated RNAs such as histone transcripts and immature mRNAs will not be captured with the poly(A)-enrichment protocol. Conversely, the alternative "ribo-minus" approach does not exclude unspliced RNAs. Do ask the sequencing facility you are going to collaborate with about the type of protocol they are using as this will inform you about the types of RNA noise that you will encounter. For more details about the different enrichment strategies and their impact, see Table S4 of Griffith et al. (2015)[†].

strand-specific sequencing If you need to distinguish overlapping transcripts, e.g., when sequencing prokaryotic transcriptomes or because the aims of the RNA-seq experiment include the identification of anti-sense transcripts, the information about which strand a fragment originated from needs to be preserved during library preparation. The most commonly used method incorporates deoxy-UTP during the synthesis of the second cDNA strand (for details see Levin et al. (2010)).

! The goal of your RNA-seq experiment will determine the most appropriate library preparation protocol. Make sure to check the most important parameters including the expected size distribution of your transcripts.

[†]`http://journals.plos.org/ploscompbiol/article/file?type=supplementary&id=info:doi/10.1371/journal.pcbi.1004393.s006`

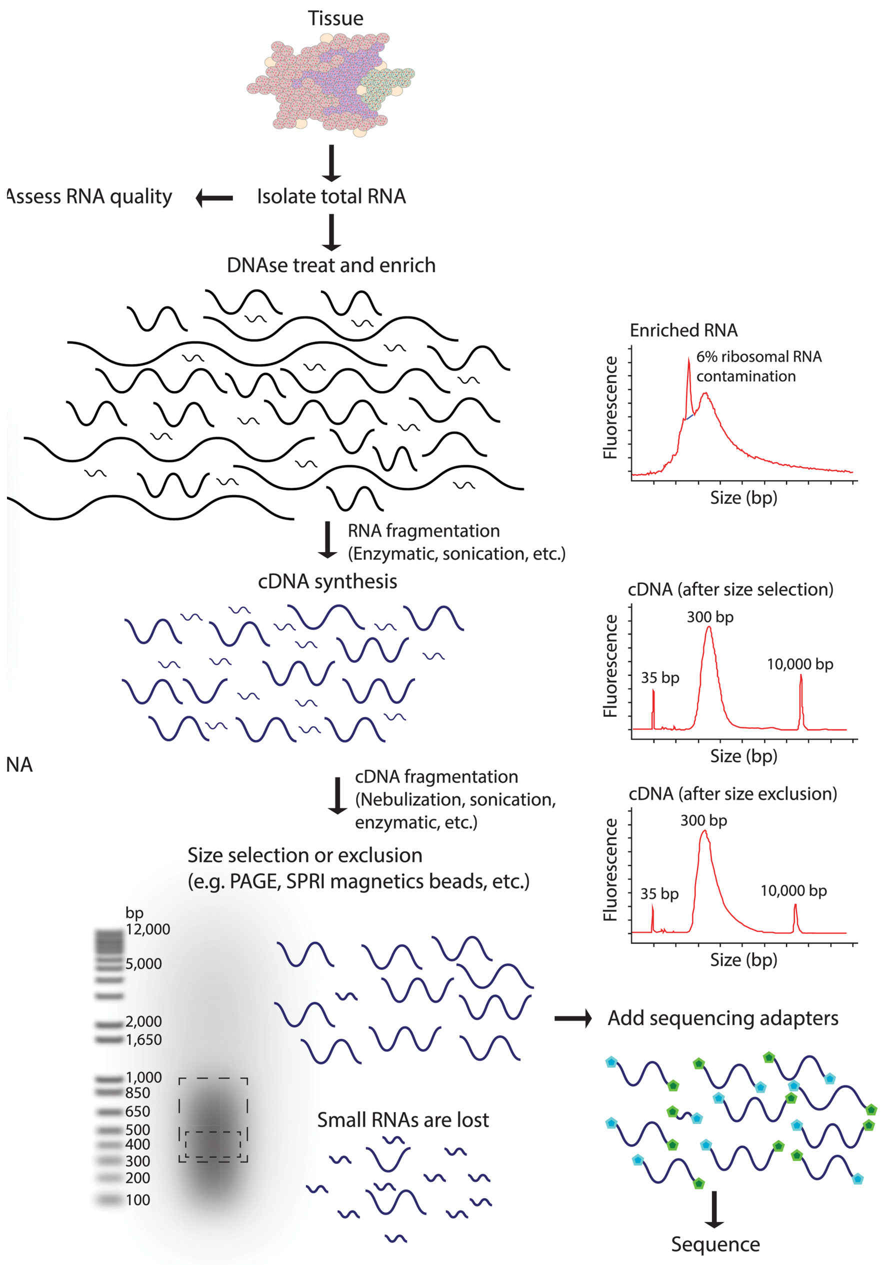


Figure 3: Size selection steps during common RNA library preparations. Typically, RNA is fragmented before or after cDNA synthesis, either via chemical (e.g. metal ion exposure), enzymatic (e.g., RNAses) or physical processes (e.g., shearing). Prior to sequencing, cDNA fragments are enriched for the size range that Illumina sequencing machines can handle best, i.e., between 150 to 1,000 bp (dashed boxes in the gel electropherogram). This means that for the vast majority of RNA-seq experiments, RNAs smaller than about 150 bp will be strongly under-represented. If you are interested in smaller RNA species, make sure that a protocol for small RNA library preparation is used. Figure taken from Griffith et al. (2015).

© 2015-2018 Applied Bioinformatics Core | Weill Cornell Medical College

1.3 Sequencing (Illumina)

After hybridization of the DNA fragments to the flowcell through means of *adapters*, each fragment is massively and clonally amplified, forming *clusters* of double-stranded DNA. This step is necessary to ensure that the sequencing signal will be strong enough to be detected unambiguously for each base of each fragment. The most commonly used Illumina sequencing protocols will only cover 50 to 100 bp of each fragment (depending on the *read* length that was chosen). The sequencing of the fragment ends is based on fluorophore-labelled dNTPs with reversible terminator elements that will become incorporated and excited by a laser one at a time and thereby enable the optical identification of single bases (Figure 4, Table 4).

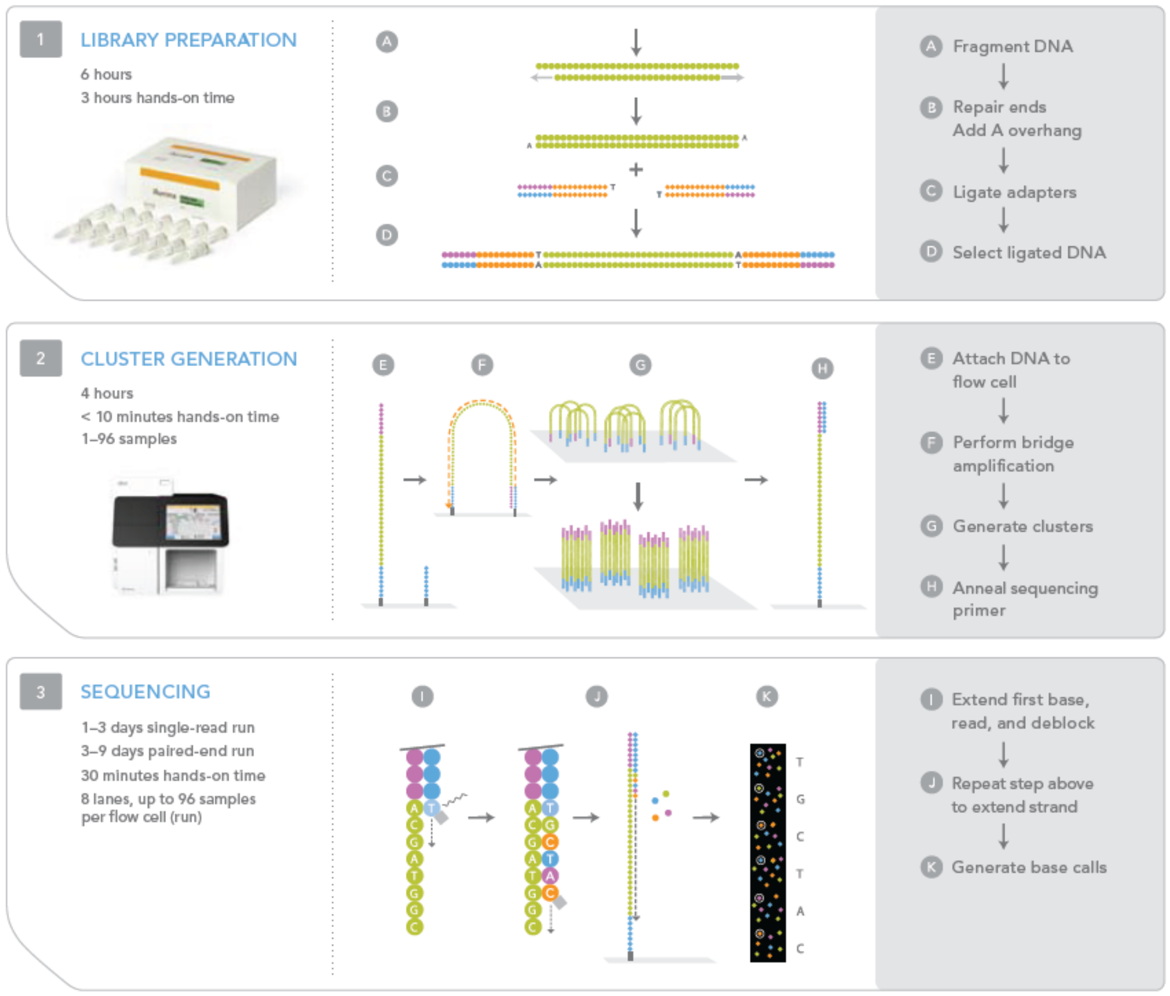


Figure 4: The different steps of sequencing with Illumina's *sequencing by synthesis* method. *Library preparation*: Adapters are ligated to each cDNA fragment to eventually attach them to the flowcell on which they are going to be sequenced. To increase the signal of the sequencing step, every fragment is first clonally amplified after the hybridization onto the flowcell (*cluster generation*). Finally, the nucleotide order of each fragment is revealed through PCR with fluorophore-labelled nucleotides: Images are taken after each round of nucleotide incorporation and bases are identified based on the recorded excitation spectra. Figure from Illumina.

Sequencing depth and coverage Technically, *coverage* refers to the number of reads being sequenced in relation to the genome size, i.e., it is an estimate of how many times each base of the genome is sequenced. For experiments based on the sequencing of the genome, the Lander-Waterman equation is commonly cited as it describes the relationship between coverage, the number of sequenced reads and the genome size:

$$coverage = \frac{read\,length * number\,of\,reads}{haploid\,genome\,length}$$

To identify sequencing errors (and possibly distinguish them from genomic variants), every base should be covered more than once. The coverage value will always be an estimate as the genome is usually not covered uniformly since, for example, euchromatic fragments tend to be overrepresented and standard PCR protocols will favor GC-rich regions and impede AT-rich regions (see Table 8 for more examples of biases that occur with Illumina sequencing platforms).

For RNA-seq, the coverage estimation has rather little practical value as the size of the transcriptome is not known as accurately as the size of the genome, and, more importantly, the per-base coverage will vary drastically between different transcripts depending, most importantly, on their expression. Thus, the number of required reads is determined by the least abundant RNA species of interest. However, it is impossible to know before sequencing how many reads are going to be needed to capture enough fragments of the most lowly expressed genes. In order to estimate the sequencing depth (= read numbers) needed for a specific RNA-seq experiment, consider the following parameters:

- guidelines from the literature/references (e.g., ENCODE (2011), Sims et al. (2014))
- type of experiment and the type of biological question
- transcriptome size and complexity (many repetitive regions present?)
- error rate of the sequencing platform

See Table 1 for recommended numbers of reads for typical RNA-seq applications. Be aware that, depending on your application, you may want to sequence deeper – consider increasing the number of reads if your goal is to:

- identify lowly expressed genes
- identify very small fold changes between different conditions
- quantify alternative splicing/different isoforms
- detect chimeric transcripts
- detect novel transcripts, transcription start and end sites
- perform *de novo* transcript assembly

Keep in mind that strongly expressed genes and residual rRNA will always account for a large fraction of all reads.

If you are interested in performing power analyses for differential gene expression detection using RNA-seq data, you can have a look at the publication and R code provided by Ching et al. (2014).

! In most cases of differential gene expression analysis, it is more important to increase the number of biological replicates than the sequencing depth of single samples (Rapaport et al., 2013; Ching et al., 2014; Liu et al., 2014; Gierliński et al., 2015).

Single read vs. paired-end Single read (SR) sequencing determines the DNA sequence of just one end of each DNA fragment.

Paired-end (PE) sequencing yields both ends of each DNA fragment. PE runs are more expensive (you are generating twice as many DNA reads as with SR), but they increase the mappability for repetitive regions and allow for easier identification of structural variations and indels. They may also increase the precision of studies investigating splicing variants or chimeric transcripts.

1.4 Experimental Design

Most RNA-seq experiments aim to identify genes whose expression varies between two or more experimental settings. This means, that during our downstream analyses, we will test every single gene whether its expression seems to change when comparing two (or more) conditions. It seems immediately obvious that

© 2015-2018 Applied Bioinformatics Core | Weill Cornell Medical College

Table 1: Recommended sequencing depths for typical RNA-seq experiments for different genome sizes (Genohub, 2015). DGE = differential gene expression, SR = single read, PE = paired-end.

	Small (bacteria)	**Intermediate (fruit fly, worm**	**Large (mouse, human)**
No. of reads for DGE (x10^6)	5 SR	10 SR	20–50 SR
No. of reads for *de novo* transcriptome assembly (x10^6)	30–65 PE	70–130 PE	100–200 PE
Read length (bp)	50	50–100	>100

comparing just one measurement per condition is not going to yield a very robust answer since gene expression may vary because of many factors (e.g., temperature, sex, time of the day), not just because of the condition of interest (e.g., genotype or drug treatment). To distinguish transcription changes caused by the condition being studied from transcription variation caused by differences between individual organisms, cell populations, or experimenters, it is important to perform RNA-seq experiments with sufficient numbers of different types of replicates (Table 2) and with a well thought-out experimental design.

> Our goal is to observe a reproducible effect that can be due only to the treatment (avoiding confounding and bias) while simultaneously measuring the variability required to estimate how much we expect the effect to differ if the measurements are repeated with similar but not identical samples (replicates). (Altman and Krzywinski, 2014)

1.4.1 Capturing enough variability

Without a somewhat realistic estimate of the variance in your system of interest, the statistical tests will have a very hard time to make accurate inferences about the gene expression differences. The problem may not (only) be a lack of results, but if you failed to capture a truly random subset of the population of interest in your experiment, the results you eventually obtain may only be representative of these four mice you happened to sacrifice on that specific Monday in that one lab you worked in at the time.

Ideally, there should be enough replicates to capture the breadth of the variability and to identify and isolate sources of noise. In practical terms, this usually translates to a number of replicates that allows to a) identify outlier samples and b) be able to remove them without losing too much information about the background variation between transcripts of the same sample type. The latter step should only be taken if there are valid reasons to believe that a certain sample might indeed be an outlier due to technical reasons (e.g., sequencing problems) or biological reasons that do not play a role for the question at hand.

Table 2: Replicate categories and types in a hypothetical mouse single-cell gene expression RNA sequencing experiment. Taken from Blainey et al. (2014).

	Replicate type	**Category**
Subjects	Colonies	Biological
	Strains	Biological
	Cohoused groups	Biological
	Gender	Biological
	Individuals	Biological
Sample preparation	Organs from sacrificed animals	Biological
	Methods for dissociating cells from tissue	Technical
	Dissociation runs from given tissue sample	Technical
	Individual cells	Biological
	RNA-seq library construction	Technical
Sequencing	Runs from the library of a given cell	Technical
	Reads from different transcript molecules	Variable
	Reads with unique molecular identifier from a given transcript molecule	Technical

Technical replicates Every experiment will have some random noise associated with protocols or equipment. Generally speaking, technical replicates are therefore repeated measurements of the same sample (Blainey et al., 2014). For RNA-seq specifically, the ENCODE consortium has defined technical replicates as *different library preparations from the same RNA sample*. They should account for batch effects from the library preparation such as reverse transcription and PCR amplification. To avoid possible lane effects (e.g., differences in the sample loading, cluster amplification, and efficiency of the sequencing reaction), it is good practice to multiplex the same sample over different lanes of the same flowcell. In most cases, technical variability introduced by the sequencing protocol is quite low and well controlled, so that technical replicates accounting for library preparation alone are rarely done – as long as you use the same protocol and the same sequencing center for all your samples.

Biological replicates There is an on-going debate over what kinds of samples represent true biological replicates, but a generally accepted definition is that biological replicates should be "parallel measurements of biologically distinct samples that capture random biological variation" (Blainey et al., 2014). Biological replicates will allow you to have a better handle on the true mean and variance of expression (of all genes in question) for the biological population of interest. The ENCODE consortium specifies that biological replicates should represent *RNA from an independent growth of cells/tissue* (ENCODE (2011)). Nevertheless, for complex experimental designs, this may mean that the distinction between biological and technical replicates depends on which sources of variation are of interest and which ones are being viewed as noise sources.

Numbers of replicates Currently, most published RNA-seq experiments contain three biological replicates. Based on one of the most exhaustive RNA-seq experiment reported to-date (48 replicates per condition), Schurch et al. (2016) recommend the use of at least six replicates per condition if the focus is on a reliable description of one condition's transcriptome or strongly changing genes between two conditions. If the goal of the experiment is to identify as many differentially expressed genes as possible (including slightly changing ones and those that are lowly expressed), as many as twelve replicates are recommended.

Always keep in mind that you are ultimately trying to draw conclusions about entire populations of cells or even organisms just by looking at very selective subsets of these. The degree of generalizability of your findings to, say, all mice of a specific strain, will strongly depend on how well you were able to capture good representatives in your experiment.

! As a general rule, the more genes with low fold changes that are to be detected, the more replicates are needed to increase the precision of the estimation of the biological variability.

Artificial RNA spike-in If it is important to you to accurately quantify *absolute* transcript concentrations, you may want to consider to use spike-ins of artificial RNA (such as the ERCC spike-in standard which consists of This set consists of 92 polyadenylated transcripts of varying lengths (250âĂŞ2,000 nucleotides) and GC-contents (5âĂŞ51%) (Jiang et al., 2011)). These RNA of known quantities can be used for the calibration of the RNA concentrations in each sample and to assess the sensitivity, coverage and linearity of your experiment, i.e., the overall *technical performance* of your experiment. The ERCC has released its own R package for analyzing spike-ins: `erccdashboard`, which is available at Bioconductor (Munro et al., 2014). Note that different spike-in controls are needed for each type of RNA, but standards are not yet available for all RNA types (ENCODE, 2011).

Spike-ins should not be be used for normalizing between different samples since they cannot account for differences in the amount of starting material, which will almost always be the case (unless you are sure you extracted RNA from the same number of cells with the same efficiency for all samples). In addition, Risso et al. (2014) (and others) demonstrated that the application of the spike-ins is not as reliable as one would hope for.

1.4.2 Avoiding bias

The main goal of a well planned experiment is to improve the precision of the answers you will eventually get. This means that you should:

1. Identify the question of interest (What is the effect you are truly after?);
2. Attempt to identify possible sources of variability (*nuisance factors*);
3. Plan the experiment in a way that reduces the effect of the expected nuisance factors;
4. Protect yourself against unknown sources of variation.

If you feel overwhelmed with the lists of nuisance factors, go back to the first step and try to prioritize. It may also make sense to start with a pilot experiment first.

The next paragraphs will give you a brief summary of typical means to come up with a suitable experimental design.

Randomization In addition to sufficient numbers of replicates, true randomization when selecting replicates and performing sample preparations can help to avoid unconscious selection bias that might be caused by subtle differences in the activity of the animals, their appearance, the growth pattern of cell lines etc. True randomization means: make the decision about any of the factors of interest by *tossing a coin* (Honaas et al., 2016)! This is fairly straight-forward when the factors are easily controllable, such as deciding which batches of cells to treat with a drug and which ones to keep as a control.

Blocking Randomization is meant to protect you against falling prey to spurious signals due to unintended batch effects. Usually, you will know about some factors that are very likely to be responsible for gene expression variation, such as sex, weight, or the cell cycle status of your cells of interest. If it is feasible to group your samples into distinct classes (or "blocks") for these known sources of variation, a blocking experimental design may make sense and will help increase statistical sensitivity. For a blocking design, you will create complete sub-experiments for each class, i.e. all conditions of interest must be present in every block. By creating these blocks in which the nuisance factor is kept constant, you will be able to detect the changes that are due to the factor of interest without having to worry about the nuisance factor. If the blocking factor accounts for a sufficient amount of sample-to-sample variation, this will increase the sensitivity of the statistical tests to detect changes of interest – there is no guarantee though! Also, keep in mind though that within each block the assignment of treatments etc. should still be randomized.

! Block what you can, randomize what you cannot.

For more general insights into good experimental design, we highly recommend Nature Methods' "Points of Significance" series by Naomi Altman and Martin Krzywinski.

?

1. What is the main advantage of stranded RNA-seq libraries?
2. What are advantages of paired-end sequencing for RNA-seq experiments?
3. How do you identify possible nuisance factors when planning an experiment?

2 Raw Data (Sequencing Reads)

Most journals require that any sequencing data related to a manuscript is deposited in a publicly accessible data base. The Sequence Read Archive (SRA) is the main repository for nucleic acid sequences and it has been growing tremendously in the past years. There are three copies of the SRA which are maintained by the NCBI, the European Bioinformatics Institute, and the DNA Databank of Japan (Figure 5), respectively.

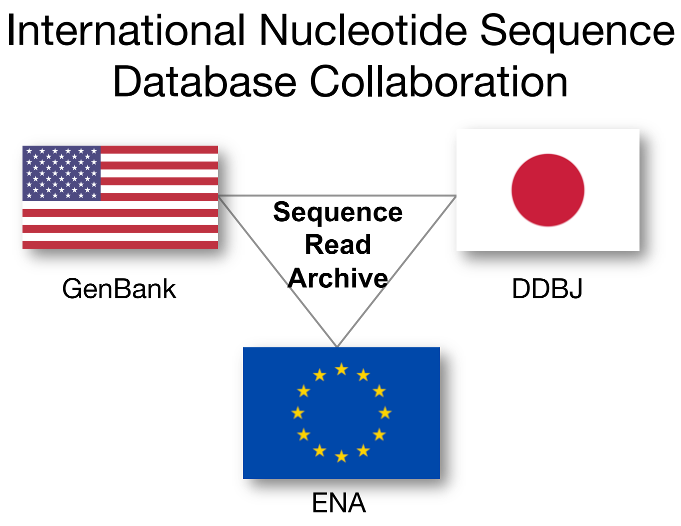


Figure 5: The Sequence Read Archive (SRA) is the largest data base of nucleic acid sequences and all three members of the International Nucleotide Sequencing Database Collaboration (GenBank, the European Nucleotide Archive, the DNA Databank of Japan) maintain instances of it.

2.1 Download from ENA

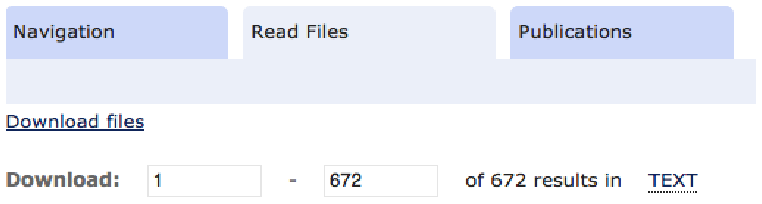
Here, we show you how to download raw sequence data from the European instance of the SRA, which can be accessed via `https://www.ebi.ac.uk/ena`. At ENA, the sequencing reads are directly available in `FASTQ` format, which will be explained below.

To download a set of `FASTQ` files:

1. Go to `https://www.ebi.ac.uk/ena`.
2. Search for the accession number of the project, e.g., ERP004763 (should be indicated in the published paper).
3. There are several ways to start the download:
   (a) Click on the link within the column "Fastq files (ftp)" and save the file of interest. Done.
   (b) If you prefer the command line, copy the link's address of the "Fastq files" column (right mouse click), go to the command line, move to the target directory, type:

```
$ wget <link copied from the ENA website>
```

   (c) If there are many samples within one project, you can download the summary of the sample information from ENA by right-clicking on "TEXT" and copying the link location.



Download: 1 - 672 of 672 results in TEXT

```
$ wget -O samples_at_ENA.txt "<LINK>" # the quotation marks are
   important
```

Once you have done this, go to the folder where you will store the data and use the 11th column of the `TEXT` file ("Fastq files (ftp)") to feed the individual `FTP URLs` of the different samples to the `wget` command:

```
$ cut -f11 samples_at_ENA.txt | xargs wget # this would download ALL
   672 samples
```

All sequencing data submitted to the SRA (i.e., with an SRA accession number) can also be retrieved through NCBI (`https://www.ncbi.nlm.nih.gov/sra`). Details about the download procedure with NCBI's SRA instance can be found here: `https://www.ncbi.nlm.nih.gov/books/NBK242621/`.

Example data Throughout the course, we will be working with sequencing reads from the most comprehensive RNA-seq dataset to date that contains mRNA from 48 replicates of two *S. cerevisiae* populations: wildtype and *snf2* knock-out mutants (Gierliński et al., 2015; Schurch et al., 2016). All 96 samples were sequenced on one flowcell (Illumina HiSeq 2000); each sample was distributed over seven lanes, which means that there are seven technical replicates per sample. The accession number for the entire data set (consisting of 7 x 2 x 48 (= 672) raw read files) is ERP004763.

? Use the information from the file `ERP004763_sample_mapping.tsv` (from `https://ndownloader.figshare.com/files/2194841`) to download all `FASTQ` files related to the biological replicates no. 1 of sample type "SNF2" as well as of sample type "WT". Try to do it via the command line and make sure to create two folders (e.g., `SNF2_rep1` and `WT_rep1`) of which each should contain seven `FASTQ` files in the end.

A simple for-loop could look like this:

```
$ for ACC_NR in ERR458493 ERR458494 ERR458495 ERR458496 ERR458497 ERR458498
  do
     egrep ${ACC_NR} ERP004763_sample_mapping.tsv | cut -f11 | xargs wget
  done
```

? Can you come up with a more generic one, e.g. without manually typing out the actual accession numbers of interest? Can you spot the vulnerabilities of the code shown above?

2.2 Storing sequencing reads: `FASTQ` format

Currently, raw reads are most commonly stored as `FASTQ` files. However, details of the file formats may vary widely depending on the sequencing platform, the lab that released the data, or the data repository. For a more comprehensive overview of possible file formats of raw sequencing data, see the NCBI's file format guide: `https://www.ncbi.nlm.nih.gov/books/NBK242622/`.

The `FASTQ` file format was derived from the simple text format for nucleic acid or protein sequences, `FASTA`. `FASTQ` bundles the sequence of every single read produced during a sequencing run together with the quality scores. `FASTQ` files are uncompressed and quite large because they contain the following information for every single sequencing read:

1. @ followed by the read ID and possibly information about the sequencing run
2. sequenced bases
3. + (perhaps followed by the read ID again, or some other description)
4. quality scores for each base of the sequence (ASCII-encoded, see below)

Again: be aware that this is not a strictly defined file format – variations do exist and may cause havoc!

Here's a real-life example snippet of a `FASTQ` file downloaded from ENA:

```
$ zcat ERR459145.fastq.gz | head
@ERR459145.1 DHKW5DQ1:219:D0PT7ACXX:2:1101:1590:2149/1
GATCGGAAGAGCGGTTCAGCAGGAATGCCGAGATCGGAAGAGCGGTTCAGC
+
@7<DBADDDBH?DHHI@DH>HHHEGHIIIGGIFFGIBFAAGAFHA'5?B@D
@ERR459145.2 DHKW5DQ1:219:D0PT7ACXX:2:1101:2652:2237/1
GCAGCATCGGCCTTTTGCTTCTCTTTGAAGGCAATGTCTTCAGGATCTAAG
+
@@;BDDEFGHHHHIIIGBHHEHCCHGCGIGGHIGHGIGIIGHIIAHIIIGI
@ERR459145.3 DHKW5DQ1:219:D0PT7ACXX:2:1101:3245:2163/1
TGCATCTGCATGATCTCAACCATGTCTAAATCCAAATTGTCAGCCTGCGCG
```

!

For **paired-end** (PE) sequencing runs, there will always be **two** FASTQ files – one for the forward reads, one for the backward reads.
Once you have downloaded the files for a PE run, make sure you understand how the origin of each read (forward or reverse read) is encoded in the read name information as some downstream analysis tools may require you to combine the two files into one.

?

1. Count the number of reads stored in a `FASTQ` file.
2. Extract just the quality scores of the first 10 reads of a `FASTQ` file.
3. Concatenate the two `FASTQ` files of a PE run.

Sequence identifier The first line of each FASTQ read snippet contains the read ID. Earlier Illumina sequencing platforms (< version 1.8) generated read IDs with the following format:

`@<machine_id>:<lane>:<tile>:<x_coord>:<y_coord>#<index>/<read_#>`

Starting from version 1.8 the sequence identifier line has the following format:

```
@<machine_id>:<run number>:<flowcell ID>:<lane>:<tile>:<x-pos>:<y-pos>
 <read>:<is filtered>:<control number>:<index sequence>
```

Base call quality scores Illumina sequencing is based on identifying the individual nucleotides by the fluorescence signal emitted upon their incorporation into the growing sequencing read (Figure 4). Once the sequencing run is complete, images taken during each DNA synthesis step are analyzed and the read clusters' fluorescence intensities are extracted and translated into the four letter code. The deduction of nucleotide sequences from the images acquired during sequencing is commonly referred to as *base calling*.

Due to the imperfect nature of the sequencing process and limitations of the optical instruments (see Table 8), base calling will always have inherent uncertainty. This is the reason why `FASTQ` files store the DNA sequence of each read together with a position-specific quality score that represents the error probability, i.e., how likely it is that an individual base call may be incorrect. The score is called Phred score, Q, which is proportional to the probability p that a base call is incorrect, where $Q = -10 * log_{10}(p)$. For example, a Phred score of 10 corresponds to one error in every ten base calls ($Q = -10 * log10(0.1)$), or 90% accuracy; a Phred score of 20 corresponds to one error in every 100 base calls, or 99% accuracy. A higher Phred score thus reflects higher confidence in the reported base.

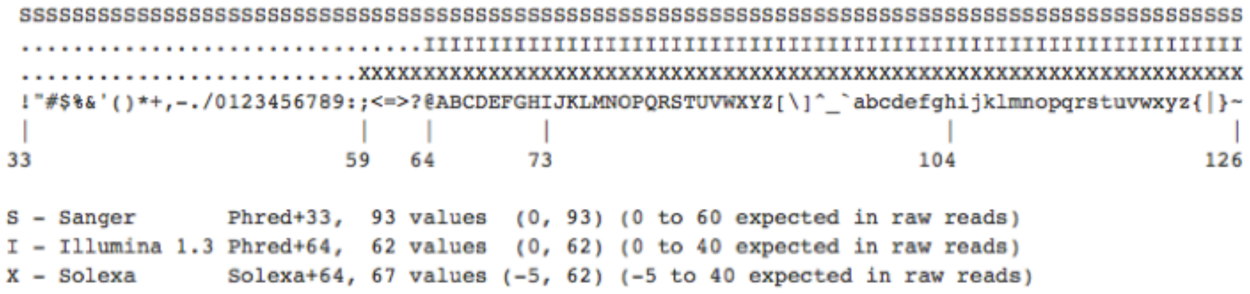
To assign each base a unique score identifier (instead of numbers of varying character length), Phred scores are typically represented as ASCII characters. At `http://ascii-code.com/` you can see which characters are assigned to what number.

© 2015-2018 Applied Bioinformatics Core | Weill Cornell Medical College

For raw reads, the range of scores will depend on the sequencing technology and the base caller used (Illumina, for example, used a tool called `Bustard`, or, more recently, `RTA`). Unfortunately, Illumina has been anything but consistent in how they a) calculated and b) ASCII-encoded the Phred score (see Table 3 and Figure 6 for the different conventions)! In addition, Illumina now allows Phred scores for base calls with as high as 45, while 41 used to be the maximum score until the HiSeq X. This may cause issues with downstream applications that expect an upper limit of 41.

Table 3: Base call quality scores are represented with the Phred range. Different Illumina (formerly Solexa) versions used different scores and ASCII offsets. Starting with Illumina format 1.8, the score now represents the standard Sanger/Phred format that is also used by other sequencing platforms and the sequencing archives. Also see Figure 6.

Description	**ASCII characters**		**Quality score**	
	Range	Offset	Type	Range
Solexa/early Illumina (1.0)	59 to 126 (; to ~)	64	Solexa	-5 to 62
Illumina 1.3+	64 to 126 (@ to ~)	64	Phred	0 to 62
Sanger standard/Illumina 1.8+	33 to 126 (! to ~)	33	Phred	0 to 93



```
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
...............................IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
..........................XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
|                         |    |        |                              |                     |
33                       59   64       73                             104                  126

S - Sanger        Phred+33,  93 values  (0, 93) (0 to 60 expected in raw reads)
I - Illumina 1.3 Phred+64,  62 values  (0, 62) (0 to 40 expected in raw reads)
X - Solexa        Solexa+64, 67 values (-5, 62) (-5 to 40 expected in raw reads)
```

Figure 6: The ASCII interpretation and ranges of the different Phred score notations used by Illumina and the original Sanger interpretation (also see Table 3. Although the Sanger format allows a theoretical score of 93, raw sequencing reads typically do not exceed a Phred score of 60. In fact, most Illumina-based sequencing will result in maximum scores of 41 to 45.

! Note that different base quality assignments exist (Table 3). Try to always make sure you know which version of the Phred score you are dealing with.

The converting of an Illumina `FASTQ` file version 1.3 (Phred+64) to version 1.8 (Phred+33) can be accomplished with the following `sed` command:

```
$ sed -e '4~4y/@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_`abcdefghi/!"#$%&'\''()
   *+,-.\/0123456789:;<=>?@ABCDEFGHIJ/' originalFile.fastq
```

?

1. Which base call is more likely to be incorrect – one with a Phred score of "#" or one with a Phred score of ";"?
2. Can you guess why the Phred scores are always transformed to ASCII with an offset of at least 33?
3. What is the baseline uncertainty that Illumina grants to its base calling algorithm?
4. How can you find out which Phred score encoding was used in a given `FASTQ` file?

2.3 Quality control of raw sequencing data

Quality controls should be done at every analysis step. Ideally, quality control should be proactive and comprehensive — see it as a chance to get to know your data, which will enable you to perform downstream analyses with (more) appropriate assumptions and parameters. Even if flaws and biases are identified, you may be able to correct those problems *in silico*.

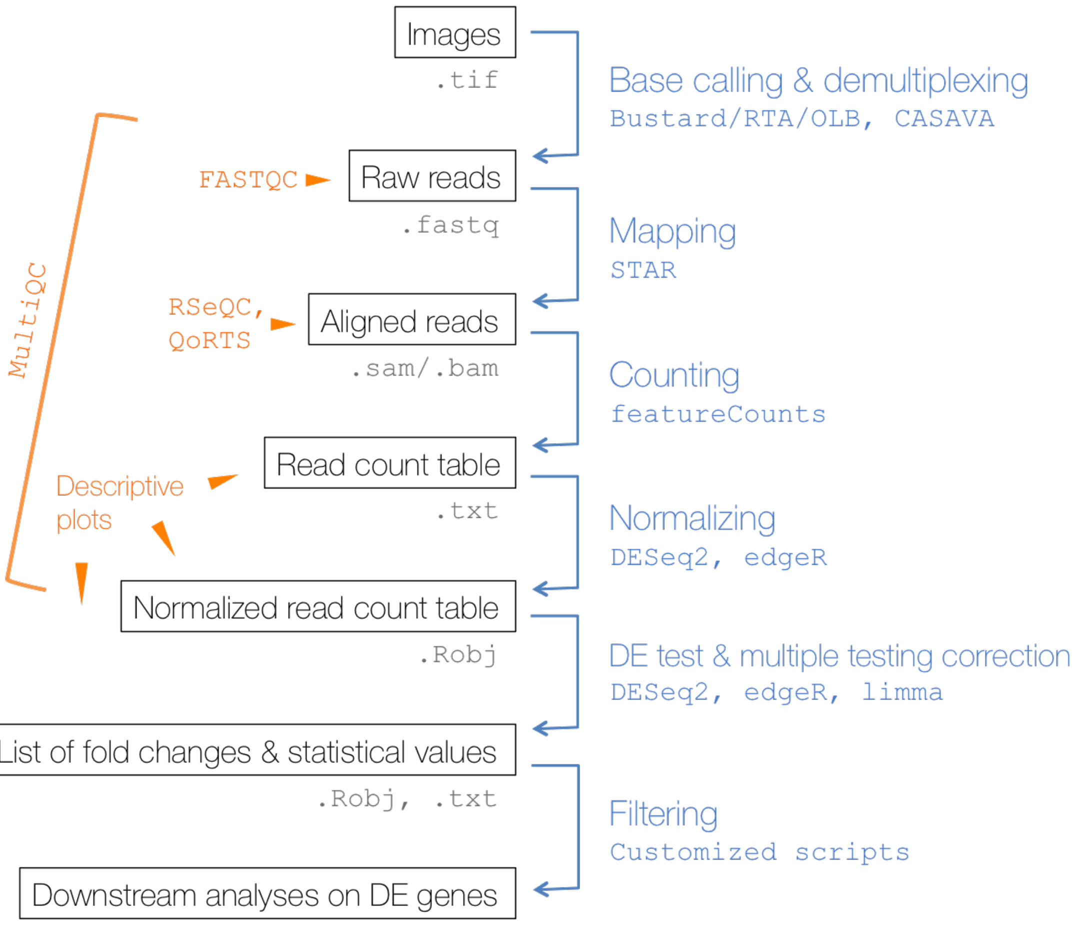


Figure 7: Typical bioinformatics workflow of differential gene expression analysis with commonly used tools (shown in blue). Tools for quality control are marked in orange (with `MultiQC` allowing the convenient combination of numerous QC results). The most commonly used file formats to store the results of each processing step are indicated in gray.

Since an analysis typically starts with the raw reads (stored in `FASTQ` files), your first step should be to check the overall quality of the sequenced reads. A poor RNA-seq run will be characterized by the presence of one or more of the following types of uninformative sequences:

- PCR duplicates**
- adapter contamination
- rRNA and tRNA reads
- unmappable reads, e.g. from contaminating nucleic acids

All but the last category of possible problems can be detected using a program called `FASTQC`. `FASTQC` is released by the Babraham Institute and can be freely downloaded at `http://www.bioinformatics.babraham.ac.uk/projects/fastqc/`. It performs multiple tests to evaluate the quality scores as well as the sequence composition of the reads stored in a given `FASTQ` file. Each test is flagged with either "pass", "warning", or "fail", depending on how far the sample deviates from a hypothetical dataset without significant

*It is impossible to distinguish whether identical reads represent PCR duplicates or independent occurrences of the exact same transcript fragment.

bias (see Table 9 for the different tests carried out by FASTQC). Keep in mind though that some sample types are *expected* to have certain biases, so not all “fail" verdicts mean that the sequencing should be repeated.

To run FASTQC, use the following command:

```
$ mkdir fastqc_results # make a folder to store the results

# run FastQC (for the course it's available in the software folder)
$ ~/mat/software/FastQC/fastqc ERR458493.fastq.gz --extract -o fastqc_results

# have a look at the results
$ ls fastqc_results/ERR458493_fastqc/
  fastqc_data.txt
  fastqc.fo
  fastqc_report.html # open this to get a quick visual impression of the
      results
  Icons/
  Images/
  summary.txt # textual summary

$ cat fastqc_results/ERR458493_fastqc/summary.txt
  PASS  Basic Statistics  ERR458493.fastq.gz
  PASS  Per base sequence quality ERR458493.fastq.gz
  FAIL  Per tile sequence quality ERR458493.fastq.gz
  PASS  Per sequence quality scores ERR458493.fastq.gz
  FAIL  Per base sequence content ERR458493.fastq.gz
  PASS  Per sequence GC content ERR458493.fastq.gz
  PASS  Per base N content  ERR458493.fastq.gz
  PASS  Sequence Length Distribution  ERR458493.fastq.gz
  WARN  Sequence Duplication Levels ERR458493.fastq.gz
  PASS  Overrepresented sequences ERR458493.fastq.gz
  PASS  Adapter Content ERR458493.fastq.gz
  WARN  Kmer Content  ERR458493.fastq.gz
```

If you ran FASTQC on more than one file, you may want to combine the plots with the brief text summary to quickly identify outlier samples. The following commands extract all test results that did not pass (grep -v PASS) and combines them with all images into a single PNG file using the montage tool. All commands are carried out for the sample names stored in files.txt (one file name per line). convert can be used to merge all PNG files into a single PDF file.

```
# extract the IDs of the individual files for WT replicate 1
$ awk '$3 == "WT" && $4 == 1 {print $1}' ERP004763_sample_mapping.tsv > files.
   txt

$ head -n3 files.txt
  ERR458493
  ERR458494
  ERR458495

$ while read ID
  do
     grep -v PASS ${ID}_fastqc/summary.txt | \
     montage txt:- ${ID}_fastqc/Images/*png \
         -tile x3 -geometry +0.1+0.1 -title ${ID} ${ID}.png
  done < files.txt

$ convert *png fastqc_summary.pdf
```

As you can see, this can be become quite cumbersome for numerous samples. Fortunately, MultiQC, a recently published (and still continuously updated) tool allows you to summarize the output of myriad QC programs (such as FastQC) in a very convenient manner (Ewels et al., 2016).

```
# run FastQC on all fastq.gz files per sample
$ for SAMPLE in WT_1 WT_2 WT_3 WT_25 # random selection of samples
  do
  mkdir fastqc_results/${SAMPLE}
  ~/mat/software/FastQC/fastqc ~/mat/precomputed/rawReads_yeast_Gierlinski/${
      SAMPLE}/*fastq.gz -o fastqc_results/${SAMPLE}
  done

# run multiqc within the fastqc_results folder
# and use the folder names (WT_1 etc.) as prefixes
# to the sample names in the final output
$ cd fastqc_results/
$ ~/mat/software/anaconda2/bin/multiqc . --dirs -o QC/

# either open the resulting html on the server
$ firefox multiqc_report.hmtl

# or send it to yourself, e.g. by downloading it on to your computer:
$ scp <username>@<IP address>:<path on the server to>/multiqc_report.html .
# if the download doesn't work, you can check if the server you're running your
     analyses on has an email tool, e.g. mutt or mailx
$ echo "QC report of raw reads" | mutt -s "MultiQC results" yourself@provider.
    com -a "fastqc_results/multiqc_report.html"
```

We highly recommend to pay `http://multiqc.info/` a visit to learn more about the capabilities of `MultiQC`.

© 2015-2018 Applied Bioinformatics Core | Weill Cornell Medical College

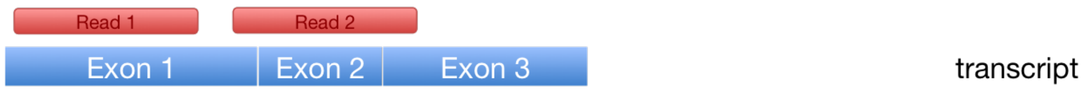
3 Read Alignment

In order to identify the transcripts that are present in a specific sample, the genomic origin of the sequenced cDNA fragments must be determined. The assignment of sequencing reads to the most likely locus of origin is called *read alignment* or *mapping* and it is a crucial step in most types of high-throughput sequencing experiments [*].

The general challenge of short read alignment is to map millions of reads accurately and in a reasonable time, despite the presence of sequencing errors, genomic variation and repetitive elements. The different alignment programs employ various strategies that are meant to speed up the process (e.g., by indexing the reference genome) and find a balance between mapping fidelity and error tolerance.

The main challenge of RNA-seq data in particular is the spliced alignment of exon-exon-spanning reads (Figure 8) and the presence of multiple different transcripts (isoforms) of the same gene. Some alignment programs tried to mitigate this problem by aligning to the transcriptome, but this approach is limited to known transcripts and thus heavily dependent on the annotation. Moreover, many reads will overlap with more than one isoform, introducing mapping ambiguity. Thus, the most popular RNA-seq alignment programs (e.g., STAR, TopHat, GSNAP; see Engström et al. (2013) for a review of RNA-seq aligners) nowadays use existing gene annotation for the placement of spliced reads in addition to attempting to identify novel splice events based on reads that cannot be aligned to the reference genome (or transcriptome). The identification of novel splice junctions is based on certain assumptions about transcript structures that may or may not be correct (e.g., most algorithms search for the most parsimonious combination of exons which might not reflect the true biological driving force of isoform generation). Additionally, lowly expressed isoforms may have very few reads that span their specific splice junctions while, conversely, splice junctions that are supported by few reads are more likely to be false positives. Therefore, novel splice junctions will show a bias towards strongly expressed genes. Until reads routinely are sequenced longer, the alignment of spliced reads will therefore remain the most prevalent problems of RNA-seq data[†]

(a) Aligning to the transcriptome



(b) Aligning to the genome

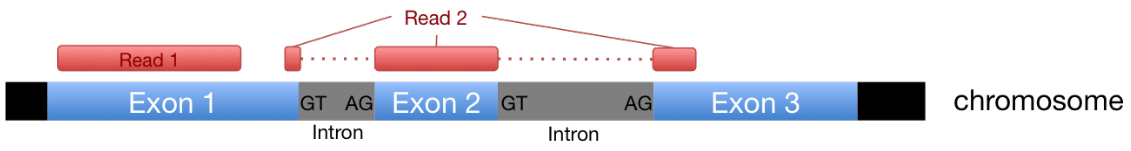


Figure 8: RNA-seq of mRNAs produces two kinds of reads: single exon reads (Read 1) and exon-exon-spanning reads (Read 2). While single exon reads can be aligned equally easily to the genome and to the transcriptome, exon-exon-spanning reads have to be split in order to be aligned properly if only the genome sequence is used as a reference (b).

3.1 Reference genomes and annotation

Genome sequences and annotation are often generated by consortia such as (mod)ENCODE, The Mouse Genome Project, The Berkeley Drosophila Genome Project, and many more. The results of these efforts can either be downloaded from individual websites set up by the respective consortia or from more comprehensive

[*] More recently, "alignment-free" methods of transcript quantification have been proposed(Patro et al., 2017; Bray et al., 2016), which will not be discussed here. In short, these tools do not determine the exact location of a read within a transcript. Instead, they compare k-mers of the reads to hash tables full of k-mers of the transcriptome and simply take note when matches occur.

[†] For comparisons of the approaches to differential isoform quantification, see Ding et al. (2017), Hooper (2014), and Su et al. (2014).

data bases such as the one hosted by the University of California, Santa Cruz (UCSC; `https://genome.ucsc.edu/`) or the European genome resource, Ensembl (`http://www.ensembl.org`).

UCSC and Ensembl try to organize, unify, and provide access to a wide range of genomes and annotation data. The advantage of downloading data from UCSC (or Ensembl) is that even if you were to work with different species, the file formats and naming conventions will be consistent (and your scripts will be more likely to work). The documentation at `https://genome.ucsc.edu/FAQ/FAQreleases.html` gives a good overview of the genomes and annotation that are available at UCSC. Unfortunately, UCSC and Ensembl differ in their naming conventions and the frequency of updates.

! Note that UCSC and Ensembl use slightly different naming conventions that can seriously affect downstream analyses. Try to stick to one source.
Always ensure you know exactly which version of a genome and annotation you are working with.

Reference sequences are usually stored in plain text `FASTA` files that can either be compressed with the generic `gzip` command or, using the tool `faToTwoBit`, into `.2bit` format.

We used the UCSC Genome Browser website to download the reference genome of yeast (go to `https://genome.ucsc.edu/`, click on "Downloads" → "Genome Data" to reach `http://hgdownload.soe.ucsc.edu/downloads.html`, where you will find an overview of all available reference genomes and the respective links).

```
# Download genome sequence of S. cerevisiae from UCSC
$ wget http://hgdownload.soe.ucsc.edu/goldenPath/sacCer3/bigZips/sacCer3.2bit

# turning compressed 2bit format into FASTA format
$ ~/mat/software/UCSCtools/twoBitToFa sacCer3.2bit sacCer3.fa

$ head sacCer3.fa
>chrI
CCACACCACACCCACACACCCACACACCACACCACACACCACACCACACC
CACACACACACATCCTAACACTACCCTAACACAGCCCTAATCTAACCCTG
GCCAACCTGTCTCTCAACTTACCCTCCATTACCCTGCCTCCACTCGTTAC
CCTGTCCCATTCAACCATACCACTCCGAACCACCATCCATCCCTCTACTT
ACTACCACTCACCCACCGTTACCCTCCAATTACCCATATCCAACCCACTG
CCACTTACCCTACCATTACCCTACCATCCACCATGACCTACTCACCATAC
TGTTCTTCTACCCACCATATTGAAACGCTAACAAATGATCGTAAATAACA
CACACGTGCTTACCCTACCACTTTATACCACCACCACATGCCATACTCAC
CCTCACTTGTATACTGATTTTACGTACGCACACGGATGCTACAGTATATA
```

3.1.1 File formats for defining genomic regions

While the reference sequence is not much more than a very long string of A/T/C/G/N, various file formats exist to store information about the location of transcription start sites, exons, introns etc. All formats agree on having one line per genomic feature, but the nature of the information contained in each row can vary strongly between the formats.

`GFF` The General Feature Format has nine required fields; the first three fields form the basic `name, start, end` tuple that allows for the identification of the location in respect to the reference genome (e.g., bases 100 to 1,000 of chromosome 1). Fields must be separated by a single TAB, but no white space. All but the final field in each feature line must contain a value; missing values should be denoted with a '.'

There are two versions of the `GFF` format in use which are similar, but not compatible:

1. `GFF` version 2 (Sanger Institute; see `http://gmod.org/wiki/GFF2` or `https://www.sanger.ac.uk/resources/software/gff/spec.html`)
2. `GFF` version 3 (Sequence Ontology Project; see `http://gmod.org/wiki/GFF3`)

`GFF2` files use the following fields:

1. **reference sequence**: coordinate system of the annotation (e.g., “Chr1”)
2. **source**: describes how the annotation was derived (e.g., the name of the annotation software)
3. **method**: annotation type (e.g., gene)
4. **start position**: 1-based integer, always less than or equal to the stop position
5. **stop position**: for zero-length features, such as insertion sites, start equals end and the implied site is to the right of the indicated base
6. **score**: e.g., sequence identity
7. **strand**: “+" for the forward strand, “-" for the reverse strand, or “." for annotations that are not stranded
8. **phase**: codon phase for annotations linked to proteins; 0, 1, or 2, indicating the frame, or the number of bases that should be removed from the beginning of this feature to reach the first base of the next codon
9. **group**: contains the class and ID of an annotation which is the logical parent of the current one (“feature is composed of")

GFF3 files (asterisk denotes difference to GFF2)

1. **reference sequence**
2. **source**
3. **type***: constrained to be either: (a) a term from the “lite" sequence ontology, SOFA; or (b) a SOFA accession number.
4. **start position**
5. **stop position**
6. **score**
7. **strand**
8. **phase**
9. **attributes***: list of feature attributes as TAG=VALUE pairs; spaces are allowed in this field, multiple TAG=VALUE pairs are separated by semicolons; the TAGS have predefined meanings:
   - ID (must be unique)
   - Name (display name)
   - Alias (secondary name)
   - Parent
   - Target (the format of the value is “target_id start end [strand]")
   - Gap (in CIGAR format)
   - Derives_from (database cross reference)
   - Ontology_term

```
# GFF-version 2
IV      curated exon    5506900 5506996 . + .   Transcript B0273.1
IV      curated exon    5506026 5506382 . + .   Transcript B0273.1
IV      curated exon    5506558 5506660 . + .   Transcript B0273.1
IV      curated exon    5506738 5506852 . + .   Transcript B0273.1

# GFF-version 3
ctg123  .  exon  1300  1500  .  +  .  ID=exon00001
ctg123  .  exon  1050  1500  .  +  .  ID=exon00002
ctg123  .  exon  3000  3902  .  +  .  ID=exon00003
ctg123  .  exon  5000  5500  .  +  .  ID=exon00004
ctg123  .  exon  7000  9000  .  +  .  ID=exon00005
```

GTF The Gene Transfer Format is based on the GFF, but is defined more strictly. (It is sometimes referred to as GFF2.5 because the first eight GTF fields are the same as GFF2, but, as for GFF3, the 9th field has been expanded into a list of attributes.) Contrary to GFF files, the TYPE VALUE pairs of GTF files are separated by one space and must end with a semi-colon (followed by exactly one space if another attribute is added afterwards):

```
# example for the 9th field of a GTF file
    gene_id "Em:U62.C22.6"; transcript_id "Em:U62.C22.6.mRNA"; exon_number 1
```

The gene_id and transcript_id values are globally unique identifiers for the genomic locus of the transcript or the same transcript itself and must be the first two attributes listed. Textual attributes should be surrounded by double quotes.

```
# GTF example
chr1   HAVANA   gene   11869 14412 . + . gene_id "ENSG00000223972.4";
    transcript_id "ENSG00000223972.4"; gene_type "pseudogene"; gene_status "
    KNOWN"; gene_name "DDX11L1"; transcript_type "pseudogene"; transcript_status
     "KNOWN"; transcript_name "DDX11L1"; level 2; havana_gene "
    OTTHUMG00000000961.2";
chr1   HAVANA   transcript   11869 14409 . + . gene_id "ENSG00000223972.4";
    transcript_id "ENST00000456328.2"; gene_type "pseudogene"; gene_status "
    KNOWN"; gene_name "DDX11L1"; transcript_type "processed_transcript";
    transcript_status "KNOWN"; transcript_name "DDX11L1-002"; level 2; tag "
    basic"; havana_gene "OTTHUMG00000000961.2"; havana_transcript "
    OTTHUMT00000362751.1";
```

More information on GTF format can be found at http://mblab.wustl.edu/GTF2.html (or, for the most recent version: http://mblab.wustl.edu/GTF22.html).

The following screenshot illustrates how you can, for example, download a GTF file of yeast transcripts from the UCSC Genome Table Browser (https://genome.ucsc.edu/cgi-bin/hgTables).

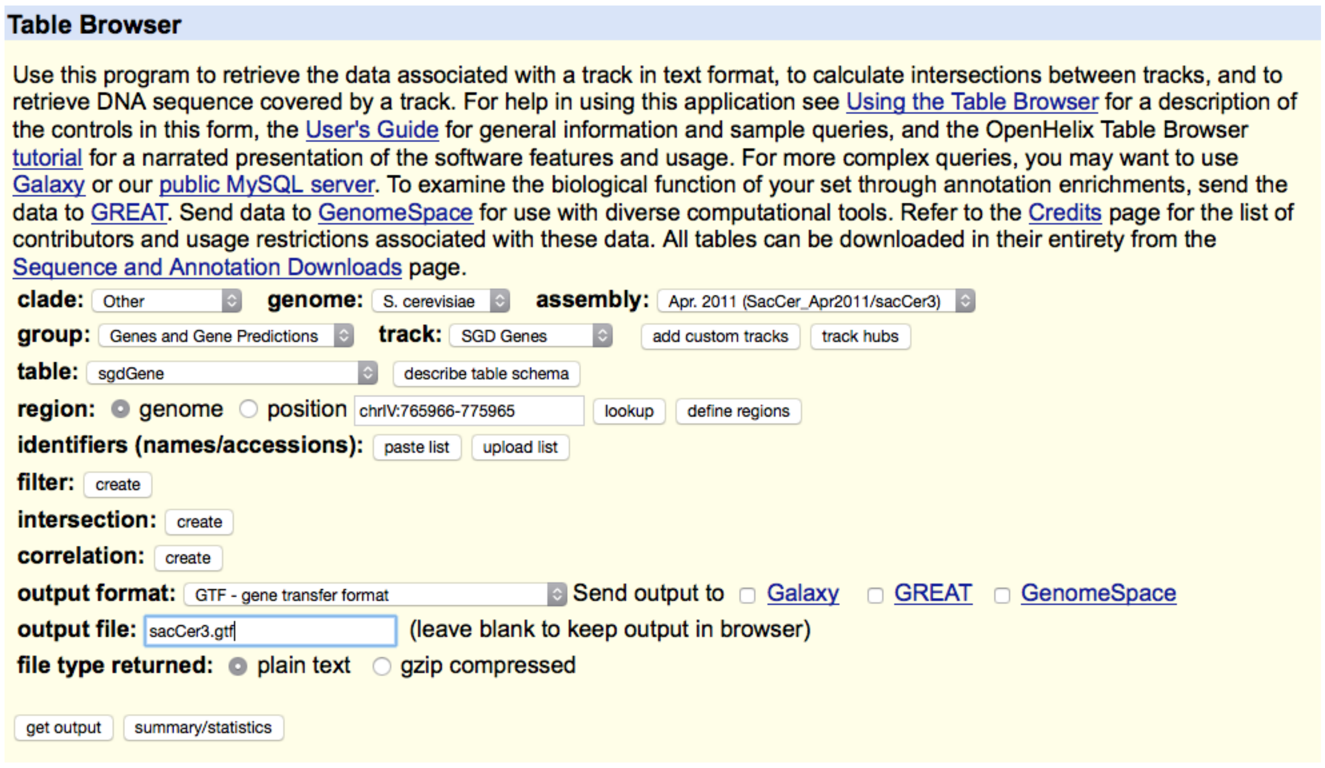


Table Browser

Use this program to retrieve the data associated with a track in text format, to calculate intersections between tracks, and to retrieve DNA sequence covered by a track. For help in using this application see Using the Table Browser for a description of the controls in this form, the User's Guide for general information and sample queries, and the OpenHelix Table Browser tutorial for a narrated presentation of the software features and usage. For more complex queries, you may want to use Galaxy or our public MySQL server. To examine the biological function of your set through annotation enrichments, send the data to GREAT. Send data to GenomeSpace for use with diverse computational tools. Refer to the Credits page for the list of contributors and usage restrictions associated with these data. All tables can be downloaded in their entirety from the Sequence and Annotation Downloads page.

clade: Other **genome:** S. cerevisiae **assembly:** Apr. 2011 (SacCer_Apr2011/sacCer3)
group: Genes and Gene Predictions **track:** SGD Genes
table: sgdGene
region: [✓ genome] position chrIV:765966-775965
identifiers (names/accessions):
filter:
intersection:
correlation:
output format: GTF - gene transfer format Send output to ☐ Galaxy ☐ GREAT ☐ GenomeSpace
output file: sacCer3.gtf (leave blank to keep output in browser)
file type returned: [✓ plain text] gzip compressed

! GTF files downloaded from the UCSC Table Browser have the same entries for gene_id and transcript_id. This can lead to problems with downstream analysis tools that expect exons of different isoforms to have the same gene_id, but different transcript_ids.

© 2015-2018 Applied Bioinformatics Core | Weill Cornell Medical College

Here's a way to get a properly formatted GTF file (i.e., with different entries for gene_name and transcript_id) of RefSeq genes using the UCSC tool genePredToGtf:

```
# first, download a table for "Genes and Gene Predictions" from the UCSC Table
   Browser indicating as the output format: "all fields from selected table"
# NOTE: this may not work for all GTF files downloaded from UCSC! genePredToGtf
    is very finicky and every organism's annotation may have been generated and
    deposited by a different person)
$ head -n1 allfields_hg19.txt
bin     name     chrom    strand  txStart txEnd    cdsStart         cdsEnd
   exonCount         exonStarts       exonEnds          score    name2    cdsStartStat
       cdsEndStatexonFrames
# remove first column and first line, feed that into genePredToGtf
$ cut -f 2- allfields_hg19.txt | sed '1d' | \
  genePredToGtf file stdin hg19_RefSeq.gtf
$ head -n1 hg19_RefSeq.gtf
chr1  stdin exon  66999639  67000051  . + . gene_id "SGIP1"; transcript_id "
   NM_032291"; exon_number "1"; exon_id "NM_032291.1"; gene_name "SGIP1";
```

BED format The BED format is the simplest way to store annotation tracks. It has three required fields (chromosome, start, end) and up to 9 optional fields (name, score, strand, thickStart, thickEnd, itemRgb, blockCount, blockSizes, blockStarts). The number of fields per line can thus vary from three to twelve, but must be consistent within a file and must obey the order, i.e. lower-numbered fields must always be populated if higher-numbered fields are used. Fields seven to twelve are only necessary if regions should be drawn in a Genome Browser with the typical appearance known for gene tracks. Note that the BED format indicates a region with 0-based start position and 1-based end position (GTF/GFF are 1-based in both positions[‡].

```
# 6-column BED file defining transcript loci
chr1  66999824  67210768  NM_032291 0 +
chr1  33546713  33586132  NM_052998 0 +
chr1  25071759  25170815  NM_013943 0 +
chr1  48998526  50489626  NM_032785 0 -
```

?

1. Which annotation data base is currently recommended for poly(A)-enriched RNA-seq data?
2. Which annotation data base would you use for RNA-seq of total RNA?
3. How many non-coding RNA transcripts does the Ensembl annotation for the human reference hg19 contain? Find out via the command line.

!

Obtaining a correctly formatted GTF file may be one of the most difficult tasks in the entire analysis! Do take this seriously and invest the time to make sure that the GTF file you are using is correctly formatted. Do not take the risk of introducing strange results (which you may not notice) that are due to formatting issues only!

[‡]See http://alternateallele.blogspot.de/2012/03/genome-coordinate-conventions.html for a very god explanation of 0- vs. 1-based interval notations)

3.2 Aligning reads using STAR

Numerous alignment programs have been published in the past (and will be published in the future), and depending on your specific project, some aligners may be preferred over others. For example, detection of structural variants and fusion transcripts will require very specific settings or a dedicated alignment tool for that particular task.

For straight-forward RNA-seq data that will be used for differential gene expression analysis, `STAR` (Dobin et al., 2013) has been shown to be very efficient and reasonably sensitive. The main caveat is the large number of putative novel splice sites that should be regarded with caution (Engström et al., 2013). The very detailed documentation of `STAR` can be found in Alex Dobin's github account: `https://github.com/alexdobin/STAR/blob/master/doc/STARmanual.pdf` and lots of advice regarding optimal parameter settings can be found in Dobin and Gingeras (2016).

Another popular aligner is `TopHat`, which is basically a sophisticated wrapper around the genomic aligner `Bowtie` (Kim et al., 2013). Generally, the specific choice of alignment tool has relatively little impact on the downstream analyses (compared to the significant impact that the choices of annotation, quantification tools, and differential expression analysis tools have; see, for example Costa-Silva et al. (2017); Everaert et al. (2017); Williams et al. (2017)). However, Ballouz et al. (2018) argue that some tools might offer a high degree of optimization for samples with specific characteristics that will not be as optimally served by using the usual top-of-the-class-tool. In any case, we strongly recommend to read the documentation of any tool you are going to use in order to tune the parameters that might be applicable to your samples.

Shown here are the example commands for the alignment to the *S. cerevisiae* genome using `STAR`.

1. **Generate genome index** This step has to be done only once per genome type (and alignment program). The index files will comprise the genome sequence, suffix arrays (i.e., tables of k-mers), chromosome names and lengths, splice junctions coordinates, and information about the genes (e.g. the strand). Therefore, the main input for this step encompasses the reference genome and an annotation file.

```
# create a directory to store the index in
$ REF_DIR=~/mat/referenceGenomes/S_cerevisiae
$ mkdir ~/STARindex

# set a variable for STAR access
$ runSTAR=~/mat/software/STAR-2.5.4b/bin/Linux_x86_64_static/STAR

# Run STAR in "genomeGenerate" mode
$ ${runSTAR} --runMode genomeGenerate \
    --genomeDir ~/STARindex \ # index will be stored there
    --genomeFastaFiles ${REF_DIR}/sacCer3.fa \ # reference genome sequence
    --sjdbGTFfile ${REF_DIR}/sacCer3.gtf \ # annotation file
    --sjdbOverhang 49 # should be read length minus 1 ; length of the
        genomic sequence around the annotated junction to be used for the
        splice junctions database
    --runThreadN 1 \ # can be used to define more processors
```

2. **Alignment** This step has to be done for each individual `FASTQ` file.
   For the particular data set used here, each sample was distributed over seven flow cell lanes, i.e., each sample has seven separate `FASTQ` files. Unlike most aligners, `STAR` will merge those files on the fly if multiple input file names are indicated. The file names must be separated by a comma without whitespaces.

```
# make a folder to store the STAR output in
$ mkdir alignment_STAR

# list fastq.gz files separated by comma without whitespaces
$ FILES=`ls -m rawReads_yeast_Gierlinski/WT_1/*fastq.gz| sed 's/ //g'`
$ FILES=`echo $FILES | sed 's/ //g'`

```

```
# execute STAR in the runMode "alignReads"
$ ${runSTAR} --genomeDir ${REF_DIR}/STARindex/ \
  --readFilesIn $FILES \
  --readFilesCommand zcat \ # necessary because of gzipped fastq files
  --outFileNamePrefix alignment_STAR/WT_1_ \
  --outFilterMultimapNmax 1 \ # only reads with 1 match in the reference
      will be returned as aligned
  --outReadsUnmapped Fastx \ # will generate an extra output file with the
      unaligned reads
  --outSAMtype BAM SortedByCoordinate \
  --twopassMode Basic \ # STAR will perform mapping, then extract novel
      junctions which will be inserted into the genome index which will
      then be used to re-map all reads
  --runThreadN 1 # can be increased if sufficient computational power is
      available
```

! The default settings or the settings shown here may not be optimal for your application (or even for this application)! Please, read the STAR manual and Dobin and Gingeras (2016) and decide which parameters are suitable for your data set!

3. `BAM` **file indexing** Most downstream applications will require a `.BAM.BAI` file together with every `BAM` file to quickly access the `BAM` files without having to load them into memory. To obtain these index files, simply run the `samtools index` command for each `BAM` file once the mapping is finished.

```
# export samtools path (for convenience)
$ export PATH=/home/classadmin/software/samtools-1.7:$PATH

# index the BAM file
$ samtools index alignment_STAR/WT_1_Aligned.sortedByCoord.out.bam
```

`STAR` has more than 100 parameters, which are all described in its manual. While the command we show above will work well for most applications (although there's one catch as you will see later on!), we strongly recommend you familiarize yourself with the `STAR` manual. The most important points are:

- handling of multi-mapped reads (e.g., how the best alignment score is assigned and the number and order in which secondary alignments are reported);
- optimization for very small genomes;
- defining the minimum and maximum intron sizes that are allowed which will basically determine how large the insertions are allowed to be that `STAR` has to include in order to make a certain read fit to a genome locus;
- handling of genomes with more than 5,000 scaffolds (usually reference genomes in a draft stage);
- using `STAR` for the detection of chimeric (fusion) and circular transcripts.

? Which `STAR` options shown above:
- ... have to be different for every sample that you map?
- ... should remain consistent for all samples of one analysis?
- ... will affect the number of reads in the final output file?

Check the Section 7.1 (Appendix) for how the alignment could be done better for the yeast data.

3.3 Storing aligned reads: `SAM/BAM` file format

The output option of `STAR` already indicates that the results of the alignment will be stored in a `SAM` or `BAM` file. The Sequence Alignment/Map (`SAM`) format is, in fact, a generic nucleotide alignment format that describes the alignment of sequencing reads (or *query sequences*) to a reference. The human readable, TAB-delimited `SAM` files can be compressed into the Binary Alignment/Map format. These `BAM` files are bigger than simply gzipped SAM files, because they have been optimized for fast random access rather than size reduction. Position-sorted BAM files can be indexed so that all reads aligning to a locus can be efficiently retrieved without loading the entire file into memory. To convert a BAM file into a SAM file, use `samtools view`:

```
# export our local installation of samtools into your PATH
$ export PATH=~/mat/software/samtools-1.7/:$PATH
$ samtools view -h WT_1_Aligned.sortedByCoord.out.bam > WT_1_Aligned.
    sortedByCoord.out.sam
```

As shown in Figure 9, SAM files typically contain a short header section and a very long alignment section where each row represents a single read alignment. The following sections will explain the SAM format in a bit more detail. For the most comprehensive and updated information go to `https://github.com/samtools/hts-specs`.

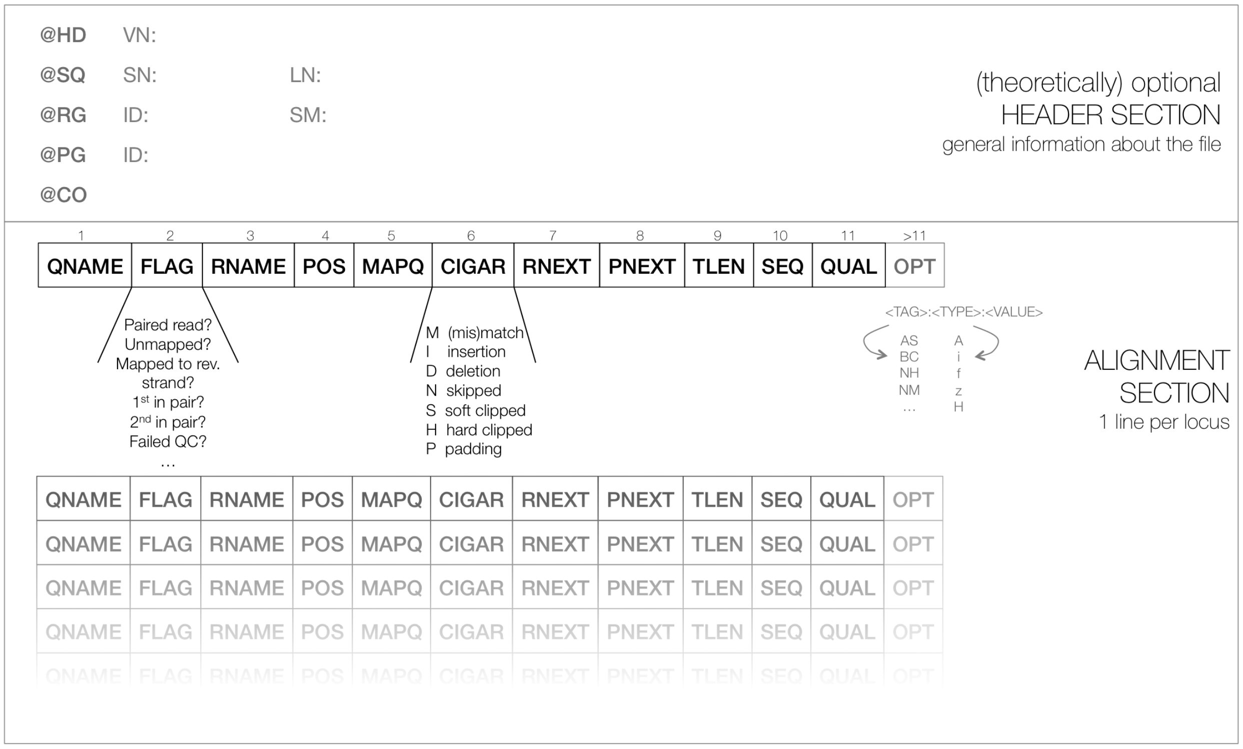


Figure 9: Schematic representation of a SAM file. Each line of the optional header section starts with "@", followed by the appropriate abbreviation (e.g., SQ for sequence dictionary which lists all chromosomes names (SN) and their lengths (LN)). See Table 10 for all possible entries and tags. The vast majority of lines within a SAM file typically correspond to read alignments where each read is described by the 11 mandatory entries (black font) and a variable number of optional fields (grey font). See Section 3.3.2 for more details.

3.3.1 The SAM file header section

The header section includes information about how the alignment was generated and stored. All lines in the header section are tab-delimited and begin with the "@" character, followed by `tag:value` pairs, where `tag` is a two-letter string that defines the content and the format of `value`. For example, the "@SQ" line in the header section contains the information about the names and lengths of the *reference sequences* to which the reads were aligned. For a hypothetical organism with three chromosomes of length 1,000 bp, the SAM header should contain the following three lines:

```
@SQ SN:chr1 LN:1000
@SQ SN:chr2 LN:1000
@SQ SN:chr3 LN:1000
```

`samtools view -H` (note the capitalized "H") can be used to retrieve just the header of a SAM or BAM file.

© 2015-2018 Applied Bioinformatics Core | Weill Cornell Medical College

The output from the following example was slightly modified for better readability. See Table 10 for more information about the entries typically stored within the header section.

```
# The default behavior of samtools view is to not show the header section.
# samtools view -h will show both header and alignment section;
# samtools view -H will return the header section only

$ samtools view -H Sample1_Aligned.sortedByCoord.out.bam
  @HD VN:1.4

  @SQ SN:chrI   LN:230218
  @SQ SN:chrII  LN:813184
  @SQ SN:chrIII LN:316620
  @SQ SN:chrIV  LN:1531933
  @SQ SN:chrV   LN:576874

  @PG ID:STAR VN:STAR_2.4.0e  CL:STAR --runThreadN 8 --genomeDir STAR-sacCer3
      --readFilesIn Lane1.fastq.gz,Lane2.fastq.gz,Lane3.fastq.gz,Lane4.fastq.gz,
      Lane5.fastq.gz,Lane6.fastq.gz,Lane7.fastq.gz --readFilesCommand zcat --
      outFileNamePrefix Sample1_ --outSAMtype BAM SortedByCoordinate --
      outSAMunmapped Within   --outFilterMultimapNmax 1

  @CO user command line: STAR --genomeDir STAR-sacCer3 --readFilesIn Lane1.
      fastq.gz,Lane2.fastq.gz,Lane3.fastq.gz,Lane4.fastq.gz,Lane5.fastq.gz,Lane6
      .fastq.gz,Lane7.fastq.gz  --readFilesCommand zcat --outFileNamePrefix
      Sample1_ --outFilterMultimapNmax 1 --outSAMunmapped Within --runThreadN 8
      --outSAMtype BAM SortedByCoordinate
```

3.3.2 The SAM file alignment section

The optional header section is followed by the alignment section where each line corresponds to one sequenced read. For each read, there are 11 mandatory fields that always appear in the same order:

`<QNAME> <FLAG> <RNAME> <POS> <MAPQ> <CIGAR> <MRNM> <MPOS> <ISIZE> <SEQ> <QUAL>`

If the corresponding information is unavailable or irrelevant, field values can be '0' or '*' (depending on the field, see Table 4), but they cannot be missing! After the 11 mandatory fields, a variable number of optional fields can be present (Figure 9).

Here's an example of one single line of a real-life SAM file:

```
ERR458493.552967  16  chrI  140 255 12M61232N37M2S  * 0 0
    CCACTCGTTCACCAGGGCCGGCGGGCTGATCACTTTATCGTGCATCTTGGC BB?
    HHJJIGHHJIGIIJJIJGIJIJJIIIGHBJJJJJJHHHHFFDDDA1+B NH:i:1  HI:i:1  AS:i:41 nM:
    i:2
```

The following table explains the format and content of each field. The FLAG, CIGAR, and the optional fields (marked in blue) are explained in more detail below.

© 2015-2018 Applied Bioinformatics Core | Weill Cornell Medical College

Table 4: Overview of the fields that are required for each row of a `SAM` file's alignment section. The number of optional fields can vary widely between different `SAM` files and even between reads within in the same file. The field types marked in blue are explained in more detail in the main text below.

Pos.	Field	Example entry	Description	NA value
1	QNAME	Read1	Query template (= read) name (PE: read pair name)	required
2	FLAG	83	Information about the read's mapping properties encoded as bit-wise flags (see next section and Table 5).	required
3	RNAME	chrI	Reference sequence name. This should match a `@SQ` line in the header.	*
4	POS	15364	1-based leftmost mapping position of the first matching base. Set as 0 for an unmapped read without coordinates.	0
5	MAPQ	30	Mapping quality of the alignment. Should be a Phred-scaled posterior probability that the position of the read is incorrect, but the value is completely dependent on the alignment program. Some tools set this to 0 if multiple alignments are found for one read.	0
6	CIGAR	51M	Detailed information about the alignment (see below).	*
7	RNEXT	=	PE reads: reference sequence name of the next read. Set to "=" if both mates are mapped to the same chromosome.	*
8	PNEXT	15535	PE reads: leftmost mapping position of the next read.	0
9	TLEN	232	PE reads: inferred template length (fragment size).	0
10	SEQ	CCA...GGC	The sequence of the aligned read on the forward strand (not including indels).	*
11	QUAL	BBH...1+B	Base quality (same as the quality string in the `FASTQ` format, but always in Sanger format [ASCII+33]).	*
12ff	OPT	NM:i:0	Optional fields (format: `<TAG>:<TYPE>:<VALUE>`; see below).	

FLAG field The FLAG field encodes various pieces of information about the individual read, which is particularly important for PE reads. It contains an integer that is generated from a sequence of Boolean bits (0, 1). This way, answers to multiple binary (Yes/No) questions can be compactly stored as a series of bits, where each of the single bits can be addressed and assigned separately.

Table 5 gives an overview of the different properties that can be encoded in the FLAG field. The developers of the SAM format and `samtools` tend to use the hexadecimal encoding as a means to refer to the different bits in their documentation. The value of the FLAG field in a given SAM file, however, will always be the decimal representation of the sum of the underlying binary values (as shown in Table 4, row 2).

Table 5: The FLAG field of SAM files stores several information about the respective read alignment in one single decimal number. The decimal number is the sum of all the answers to the Yes/No questions associated with each binary bit. The hexadecimal representation is used to refer to the individual bits (questions).

Binary (Decimal)	**Hex**	**Description**
00000000001 (1)	0x1	Is the read paired?
00000000010 (2)	0x2	Are both reads in a pair mapped “properly" (i.e., in the correct orientation with respect to one another)?
00000000100 (4)	0x4	Is the read itself unmapped?
00000001000 (8)	0x8	Is the mate read unmapped?
00000010000 (16)	0x10	Has the read been mapped to the reverse strand?
00000100000 (32)	0x20	Has the mate read been mapped to the reverse strand?
00001000000 (64)	0x40	Is the read the first read in a pair?
00010000000 (128)	0x80	Is the read the second read in a pair?
00100000000 (256)	0x100	Is the alignment not primary? (A read with split matches may have multiple primary alignment records.)
01000000000 (512)	0x200	Does the read fail platform/vendor quality checks?
10000000000 (1024)	0x400	Is the read a PCR or optical duplicate?

A bit is set if the corresponding state is true. For example, if a read is paired, `0x1` will be set, returning the decimal value of 1. Therefore, all FLAG values associated with paired reads must be uneven decimal numbers. Conversely, if the `0x1` bit is unset (= read is not paired), no assumptions can be made about `0x2`, `0x8`, `0x20`, `0x40` and `0x80`.

In a run with single reads, the flags you will most commonly see are:

- 0: This read has been mapped to the forward strand. (None of the bit-wise flags have been set.)
- 4: The read is unmapped (`0x4` is set).
- 16: The read is mapped to the reverse strand (`0x10` is set).

(`0x100`, `0x200` and `0x400` are not used by most aligners, but could, in principle be set for single reads.)

Some common FLAG values that you may see in a PE experiment include:

69	(= 1 + 4 + 64)	The read is paired, is the first read in the pair, and is unmapped.
77	(= 1 + 4 + 8 + 64)	The read is paired, is the first read in the pair, both are unmapped.
83	(= 1 + 2 + 16 + 64)	The read is paired, mapped in a proper pair, is the first read in the pair, and it is mapped to the reverse strand.
99	(= 1 + 2 + 32 + 64)	The read is paired, mapped in a proper pair, is the first read in the pair, and its mate is mapped to the reverse strand.
133	(= 1 + 4 + 128)	The read is paired, is the second read in the pair, and it is unmapped.
137	(= 1 + 8 + 128)	The read is paired, is the second read in the pair, and it is mapped while its mate is not.
141	(= 1 + 4 + 8 + 128)	The read is paired, is the second read in the pair, but both are unmapped.
147	(= 1 + 2 + 16 + 128)	The read is paired, mapped in a proper pair, is the second read in the pair, and mapped to the reverse strand.
163	(= 1 + 2 + 32 + 128)	The read is paired, mapped in a proper pair, is the second read in the pair, and its mate is mapped to the reverse strand.

Note that the strand information of the `FLAG` field (`0x10`) does not necessarily indicate the strand of the original transcript. Unless a strand-specific RNA-seq library protocol was used, this only tells you which strand of the ds-cDNA fragment was sequenced.

A useful website for quickly translating the `FLAG` integers into plain English explanations like the ones shown above is: `https://broadinstitute.github.io/picard/explain-flags.html`

?

1. How can you retrieve just the alignment section of a `BAM` file?
2. What does a `MAPQ` value of 20 mean?
3. What does a `FLAG` value of 2 mean?
4. Would you be happy or sad if your paired-end read alignments all had `FLAG` values of 77 or 141?
5. Your favorite read pair has `FLAG` values of 153 and 69. Which read aligned to the forward strand of the reference?

CIGAR [Concise Idiosyncratic Gapped Alignment Report] String The sixth field of a `SAM` file contains a so-called `CIGAR` string indicating which *operations* were necessary to map the read to the reference sequence at that particular locus.

The following operations are defined in `CIGAR` format (also see Figure 10):

`M`	Alignment (can be a sequence match or mismatch!)
`I`	Insertion in the read compared to the reference
`D`	Deletion in the read compared to the reference
`N`	Skipped region from the reference. For mRNA-to-genome alignments, an N operation represents an intron. For other types of alignments, the interpretation of `N` is not defined.
`S`	Soft clipping (clipped sequences are present in read); `S` may only have `H` operations between them and the ends of the string
`H`	Hard clipping (clipped sequences are NOT present in the alignment record); can only be present as the first and/or last operation
`P`	Padding (silent deletion from padded reference)
`=`	Sequence match (not widely used)
`X`	Sequence mismatch (not widely used)

The sum of lengths of the `M`, `I`, `S`, `=`, `X` operations must equal the length of the read.

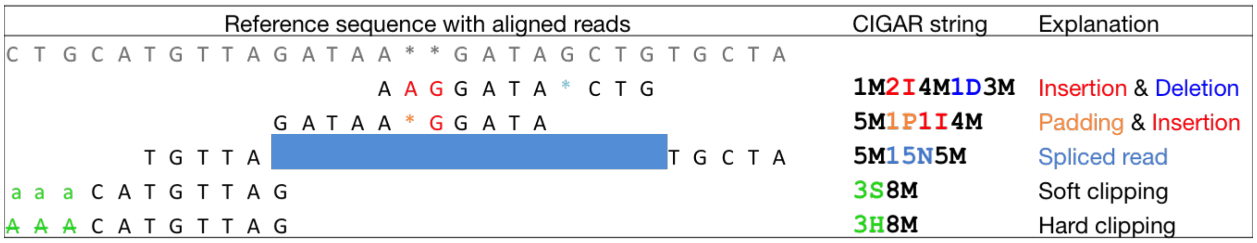


Figure 10: Image based on a figure from Li et al. (2009).

OPT field(s) Following the eleven mandatory SAM file fields, the optional fields are presented as key-value pairs in the format of `<TAG>:<TYPE>:<VALUE>`, where `TYPE` is one of:

`A`	Character
`i`	Integer
`f`	Float number
`Z`	String
`H`	Hex string

The information stored in these optional fields will vary widely depending on the mapper and new tags can be added freely. In addition, reads within the same SAM file may have different numbers of optional fields, depending on the program that generated the SAM file. Commonly used optional tags include:

`AS:i`	Alignment score
`BC:Z`	Barcode sequence
`HI:i`	Match is *i*-th hit to the read
`NH:i`	Number of reported alignments for the query sequence
`NM:i`	Edit distance of the query to the reference
`MD:Z`	String that contains the exact positions of mismatches (should complement the `CIGAR` string)
`RG:Z`	Read group (should match the entry after `ID` if `@RG` is present in the header.

Thus, for example, we can use the `NM:i:0` tag to select only those reads which map perfectly to the reference (i.e., have no mismatches).

While the optional fields listed above are fairly standardized, tags that begin with `X`, `Y`, and `Z` are reserved for particularly free usage and will never be part of the official SAM file format specifications. `XS`, for example, is used by `TopHat` to encode the strand information (e.g., `XS:A:+`) while `Bowtie2` and `BWA` use `XS:i:` for reads with multiple alignments to store the alignment score for the next-best-scoring alignment (e.g., `XS:i:30`).

3.3.3 Manipulating SAM/BAM files

As indicated above, `samtools` is a powerful suite of tools designed to interact with SAM and BAM files (Li et al., 2009).

```
# return a peek into a SAM or BAM file (note that a SAM file can also easily be
    inspected using the basic UNIX commands for any text file, such as cat,
   head, less etc.)
$ samtools view InFile.bam | head

# turn a BAM file into the human-readable SAM format (including the header)
$ samtools view -h InFile.bam > InFile.sam

# compress a SAM file into BAM format (-Sb is equivalent to -S -b)
$ samtools view -Sb InFile.sam > OutFile.bam

# generate an index for a BAM file (needed for many downstream tools)
$ samtools index InFile.bam
```

To see all the operations that can be done using `samtools`, type `samtools -help`.

The myriad information stored within the alignment files allow you to focus on virtually any subset of read alignments that you may be interested in. The `samtools view` tool has many options that directly interpret some of the mandatory fields of its alignment section (Table 4), such as the mapping quality, the location and the `FLAG` field values.

```
# get only unmapped reads
$ samtools view -h \ # show header
          -b \ # output a BAM file
          -f 4 \ # include only reads where the 0x4 bit is set
          Aligned.sortedByCoord.out.bam > unmapped_reads.bam

# get only mapped reads
$ samtools view -hb -F 4 \ # include only reads where the 0x4 bit is NOT set
          Aligned.sortedByCoord.out.bam > mapped_reads.bam

# skip read alignments with mapping quality below 20
$ samtools view -h -b -q 20 Aligned.sortedByCoord.out.bam > high_mapq_reads.bam
```

If you would like to filter an alignment file based on any of the optional tags, you will have to resort to means outside `samtools`. Looking for exact matches using `grep` can be particularly helpful here, but you should make sure that you make the regular expression search as stringent as possible.

! The number of optional `SAM`/`BAM` fields, their value types and the information stored within them completely depend on the alignment program and can thus vary substantially. Before you do any filtering on any flag, make sure you know how the aligner generated that value.

Here is an example for **retrieving reads with only one alignment** (aka uniquely aligned reads), which might be useful if `STAR` was not run with `-outFilterMultimapNmax 1`:

```
# STAR uses the NH:i tag to record the number of alignments found for a read
# NH:1 => 1 alignment; NH:2 => 2 alignments etc.
$ samtools view -h Aligned.sortedByCoord.out.bam | \ # decompress the BAM file
    egrep "^@\|\bNH:i:1\b" | \ # lines with either @ at the beginning of the
        line (= header) or exact matches of NH:i:1 are returned
    samtools view -S -b - > uniquely_aligned_reads.bam # turn the SAM file lines
         from stdin into a BAM file, - indicates standard input for samtools
```

To filter out **reads with insert sizes greater than 1000 bp**, one could make use of the `CIGAR` string. The following example assume that the alignment program indicated large insertions with the `N` operator (see Section 3.3.2) – this may not be true for all aligners!

```
# for the sake of simplicity, let's work on the SAM file:
$ samtools view -h WT_1_Aligned.sortedByCoord.out.bam > WT_1_Aligned.
    sortedByCoord.out.sam

# here's an example using grep, excluding lines with at least four digits
    followed by N
$ egrep -v "[0-9][0-9][0-9][0-9]N" WT_1_Aligned.sortedByCoord.out.sam >
    smallInsert_reads.sam

# awk can be used to match a regex within a specified column
$ awk '!($6 ~ /[0-9][0-9][0-9][0-9]N/) {print $0}' WT_1_Aligned.sortedByCoord.
    out.sam > smallInsert_reads.sam
```

To retrieve **intron-spanning reads**, the commands will be similar:

```
# egrep allows for nicer regex syntax than grep
$ egrep "(^@|[0-9]+M[0-9]+N[0-9]+M)" WT_1_Aligned.sortedByCoord.out.sam >
    intron-spanning_reads.sam

# the same result achieved with awk
$ awk '$1 ~ /^@/ || $6 ~ /[0-9]+M[0-9]+N[0-9]+M/ {print $0}' WT_1_Aligned.
    sortedByCoord.out.sam > intron-spanning_reads.sam
```

?

1. How can you extract all reads that were aligned to the reverse strand?

2. Does it make sense to filter the `BAM` files generated by `STAR` using the mapping quality filter as shown above, i.e., do you find any differences after filtering with `-q 40`?

3.4 Quality control of aligned reads

Once the reads have been aligned, the following properties should be assessed before downstream analyses are started:

- Could most reads be aligned?
- Are there any obvious biases of the read distributions?
- Are the replicate samples as similar to each other as expected?

3.4.1 Basic alignment assessments

There are numerous ways to do basic checks of the alignment success. An alignment of RNA-seq reads is usually considered to have succeeded if the mapping rate is >70%.

The very first QC of aligned reads should be to generally check the aligner's output. The `STAR` and `samtools index` commands in Section 3.2 generate the following files:

`*Aligned.sortedByCoord.out.bam`	information about the genomic loci of each read incl. its sequence
`*Log.final.out`	alignment statistics
`*Log.out`	commands, parameters, and files used
`*Log.progress.out`	elapsed time
`*SJ.out.tab`	genomic loci where splice junctions were detected and the number of reads overlapping with them
`*Unmapped.out.mate1`	text file with unmapped reads (similar to original `fastq` file)

Information about the individual output files are given in the `STAR` manual which you can find in the program's directory (e.g., STAR-STAR_2.5.4b/doc/STARmanual.pdf) or online (`https://github.com/alexdobin/STAR/blob/master/doc/STARmanual.pdf`).

?

- Which `STAR` output file will you need most for your downstream analyses?
- How can you decrease the size of the `*out.mate1` files? What format do they have?
- Which optional `SAM` fields does `STAR` add and what do they represent?

Most aligners will return a summary of the basic stats of the aligned reads, such as the number of mapped reads. For `STAR`, the information is stored in `*Log.final.out`.

© 2015-2018 Applied Bioinformatics Core | Weill Cornell Medical College

```
$ cat WT_1_Log.final.out
                                 Started job on | Jul 24 17:53:18
                             Started mapping on | Jul 24 17:53:22
                                    Finished on | Jul 24 17:53:51
       Mapping speed, Million of reads per hour | 870.78

                          Number of input reads | 7014609
                      Average input read length | 51
                                    UNIQUE READS:
                   Uniquely mapped reads number | 6012470
                        Uniquely mapped reads % | 85.71%
                          Average mapped length | 50.73
                       Number of splices: Total | 50315
            Number of splices: Annotated (sjdb) | 47843
                       Number of splices: GT/AG | 49812
                       Number of splices: GC/AG | 65
                       Number of splices: AT/AC | 7
               Number of splices: Non-canonical | 431
                      Mismatch rate per base, % | 0.36%
                         Deletion rate per base | 0.00%
                        Deletion average length | 1.37
                        Insertion rate per base | 0.00%
                       Insertion average length | 1.04
                              MULTI-MAPPING READS:
        Number of reads mapped to multiple loci | 0
             % of reads mapped to multiple loci | 0.00%
        Number of reads mapped to too many loci | 796537
             % of reads mapped to too many loci | 11.36%
                                  UNMAPPED READS:
       % of reads unmapped: too many mismatches | 0.00%
                 % of reads unmapped: too short | 2.90%
                     % of reads unmapped: other | 0.04%
```

The number of *uniquely mapped reads* is usually the most important number. If you are handling more than two BAM files, it will certainly be worthwhile to visualize the alignment rate for all files, e.g., using MultiQC or your own, customized routine in R (Figure 11).

In addition to the log files generated by the mapping program, there are numerous ways to obtain information about the numbers and kinds of reads stored in a BAM file, e.g., using samtools or RSeQC (see below). The simplest approach to finding out the number of alignments within a BAM file is to do a line count.

```
# pseudocode
$ samtools view Aligned.sortedByCoord.out.bam | wc -l
```

Note that if unmapped reads are present in the BAM file, these will also be counted, as well as multiple instances of the same read mapped to different locations if multi-mapped reads were kept. It is therefore more informative to run additional tools that will indicate the counts for specific FLAG values, too.

samtools flagstat This tool assesses the information from the FLAG field (see Section 3.3.2) and prints a summary report to the terminal.

```
$  ~/mat/software/samtools-1.5/samtools flagstat /zenodotus/abc/store/courses
    /2016_rnaseq/additionalExamples/alignment/human_samples/UHR-RIN0_Aligned.
    sortedByCoord.out.bam
66889956 + 2593364 in total (QC-passed reads + QC-failed reads)
0 + 0 secondary
0 + 0 supplementary
0 + 0 duplicates
66889956 + 2593364 mapped (100.00% : 100.00%)
0 + 0 paired in sequencing
```

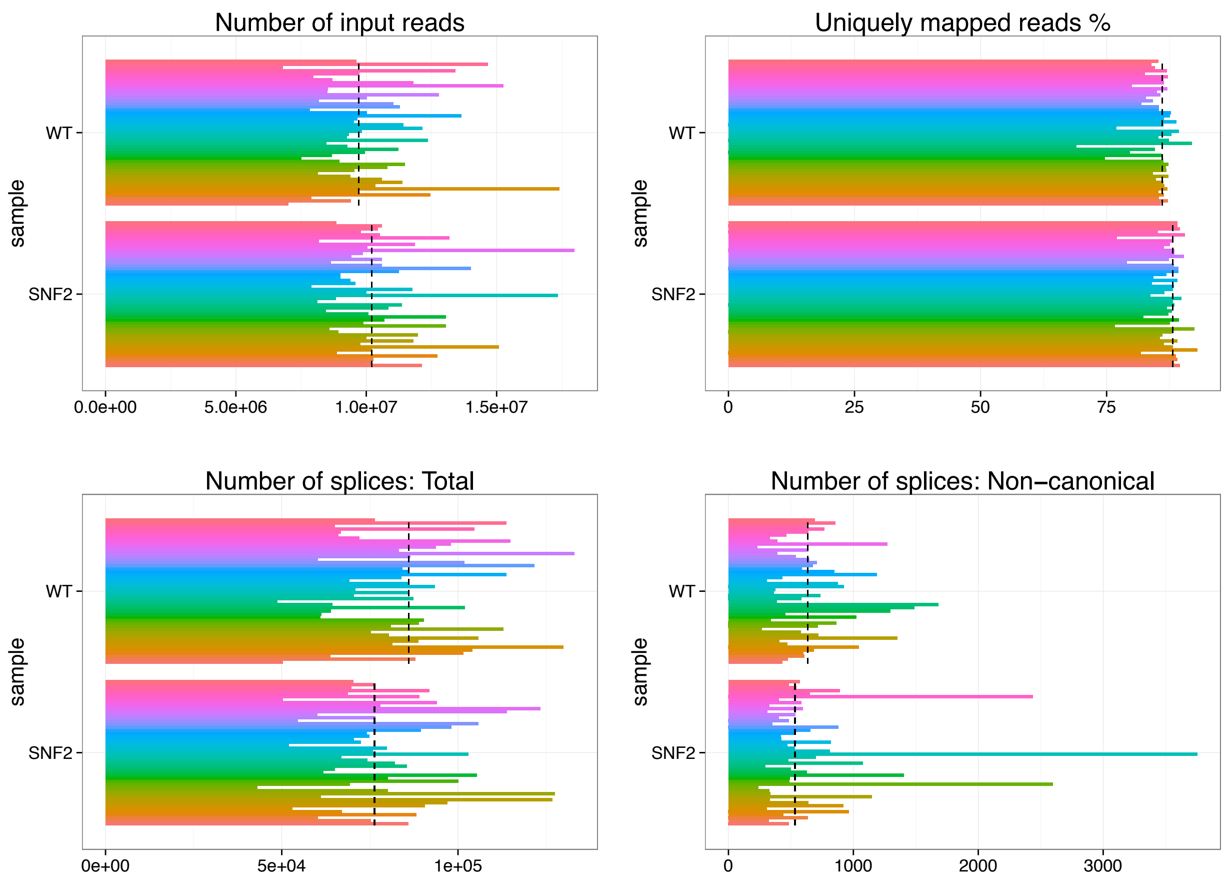


Figure 11: Graphical summary of STAR's log files for 96 samples. The individual colors represent the distinct samples, the dashed lines indicate the median values across all samples of the same condition (WT or SNF2). For details of the code underlying these figures, see `https://github.com/friedue/course_RNA-seq2015` *rightarrow* `01_Alignment_visualizeSTARresults.pdf`.

```
0 + 0 read1
0 + 0 read2
0 + 0 properly paired (N/A : N/A)
0 + 0 with itself and mate mapped
0 + 0 singletons (N/A : N/A)
0 + 0 with mate mapped to a different chr
0 + 0 with mate mapped to a different chr (mapQ>=5)
```

RSeQC's `bam_stat.py` `RSeQC` is a Python- and R-based suite of tools for various quality controls and visualizations, some of which are specific for RNA-seq experiments (Wang et al., 2012). See Table 11 for the list of all currently available scripts. Although `RSeQC` is one of the most popular tools for RNA-seq quality control, a recent publication revealed several bugs in the code of `RSeQC` (Hartley and Mullikin, 2015).

For basic alignment stats, one can use the `bam_stat.py` script:

```
# RSeQC is based on Python; add the anaconda installation of Python to your
    PATH
$ export PATH=/home/classadmin/software/anaconda2/bin/:$PATH

# now, all RSeQC scripts are immediately accessible and you can, for example,
    run bam_stat.py
$ bam_stat.py -i WT_1_Aligned.sortedByCoord.out.bam

```

```
#==================================================
#All numbers are READ count
#==================================================

Total records:                          6012470

QC failed:                              0
Optical/PCR duplicate:                  0
Non primary hits                        0
Unmapped reads:                         0
mapq < mapq_cut (non-unique):           0

mapq >= mapq_cut (unique):              6012470
Read-1:                                 0
Read-2:                                 0
Reads map to '+':                       3014735
Reads map to '-':                       2997735
Non-splice reads:                       5962195
Splice reads:                           50275
Reads mapped in proper pairs:           0
Proper-paired reads map to different chrom:0
```

If you want to add the results of `samtools flagstat` and `RSeQC's bam_stat.py` to a `MultiQC` report, capture the output that is normally printed to screen in reasonably named files.

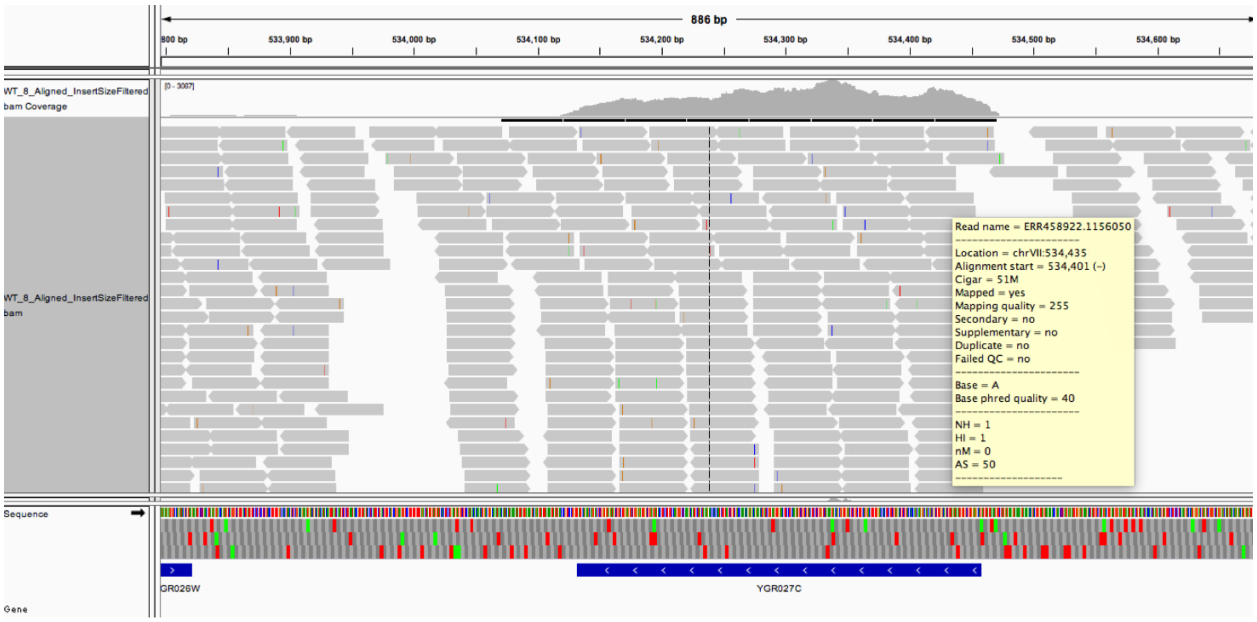
```
$ bam_stat.py -i WT_1_Aligned.sortedByCoord.out.bam > bam_stat_WT_1.txt
$ samtools flagstat WT_1_Aligned.sortedByCoord.out.bam > flagstat_WT_1.txt
```

Visualization of aligned reads It is always a good idea to visually check the results, i.e., ensure the reads align to the expected regions, preferably without too many mismatches. Here, Genome Browsers come in handy. Different research groups have released different Genome Browsers, and the most well-known browsers are probably those from Ensembl and UCSC. There are some reasons why one may not want to use these web-based options (e.g., HIPAA-protected data or lack of bandwidth to upload all data), and rather resort to stand-alone Genome Browsers (see `https://en.wikipedia.org/wiki/Genome_browser` for an overview).

We are going to use the Broad Institute's Integrative Genomics Viewer (IGV) that can be downloaded after a quick registration with an academic email address from `https://www.broadinstitute.org/software/igv/download`. It requires an up-to-date Java installation.

The IGV Genome Browser can display numerous file formats, e.g., indexed (!) `BAM` files[§] with aligned reads and `BED` files with information about genomic loci (such as genes). The following IGV snapshot (in IGV, go to "File", then "Save image") shows the region surrounding an arbitrarily chosen yeast gene (blue box) and the reads aligned to it (grey arrows).

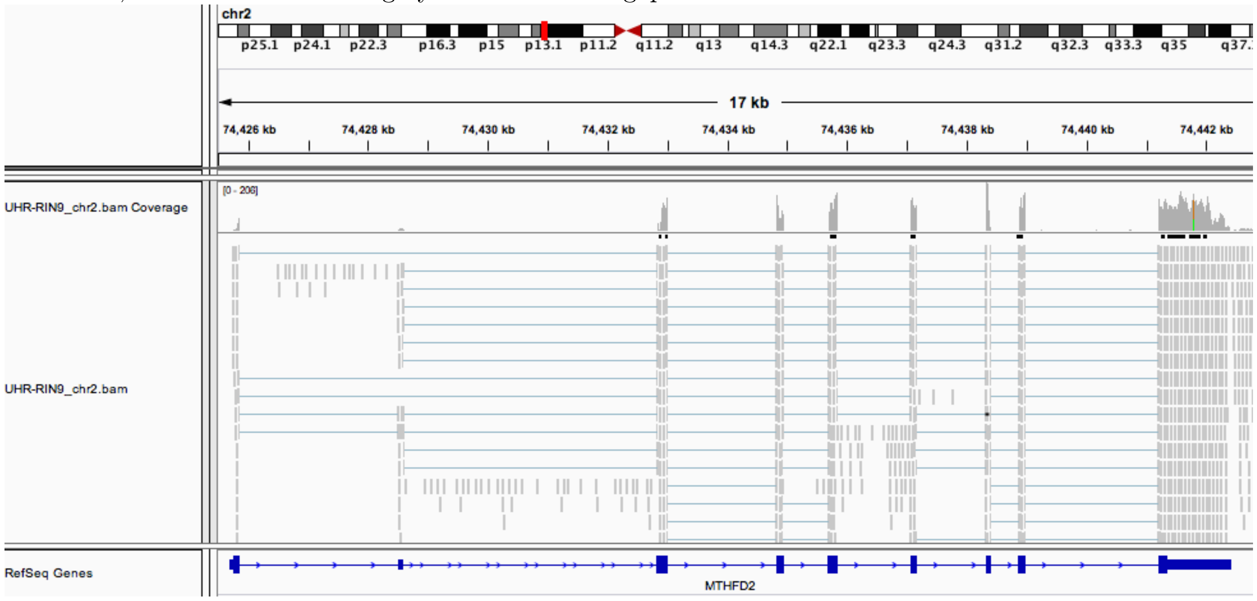
[§]That means, the `.bam` file should have a `bam.bai` file with the same base name in the same folder.



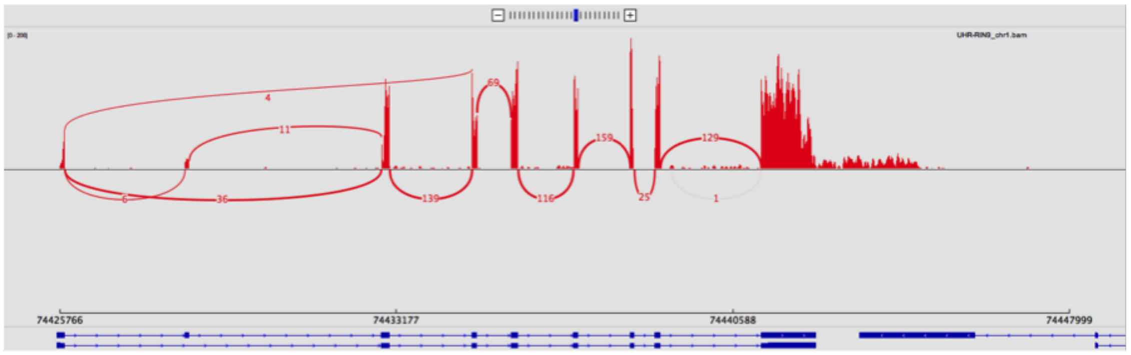
```
Read name = ERR458922.1156050
----------------------
Location = chrVII:534,435
Alignment start = 534,401 (-)
Cigar = 51M
Mapped = yes
Mapping quality = 255
Secondary = no
Supplementary = no
Duplicate = no
Failed QC = no
----------------------
Base = A
Base phred quality = 40
----------------------
NH = 1
HI = 1
nM = 0
AS = 50
-------------------
```

On top of the read alignment display, IGV also produces a coverage line that allows for quick identification of highly covered regions. Blue lines within the reads indicate insertions with respect to the reference genome, red lines indicate deletions. Since yeast genes are often intron-less, the reads can be aligned without gaps.

Human genes, however, tend to have multiple introns, which means that exon-exon-spanning reads must be aligned with often lengthy gaps within them (Figure 8). Examples of this can be seen in the following IGV screenshot, where the horizontal grey lines indicate a gap within a read:



If one is interested in the splice junctions of a particular gene, IGV can generate Sashimi plots (Katz et al., 2015): right-click on the track that contains the `BAM` file of interest and select “Sashimi plot". The result will look like this:



In the Sashimi plot, bar graphs indicate read coverage, arcs indicate splice junctions, and numbers represent the number of reads that contain the respective splice junction. Bear in mind that while the IGV-Sashimi plots are great because they allow you to interactively explore exon usage, relying on the simple read counts may be treacherous – simple differences in sequencing depth (i.e., the total number of reads per sample) can lead to perceived differences in read counts. If you want read counts normalized per million reads and adjusted for transcript length, you will have to resort to the standalone version of Sashimi (`http://miso.readthedocs.io/en/fastmiso/sashimi.html`).

3.4.2 Bias identification

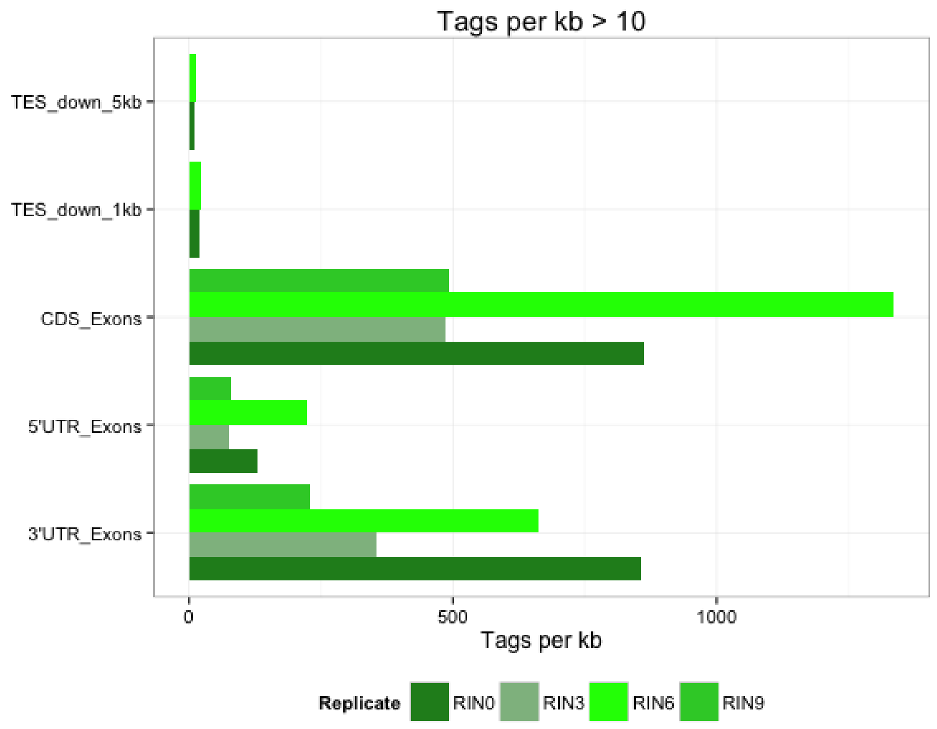
Typical biases of RNA-seq experiments include:

- **Intron coverage**: if many reads align to introns, this is indicative of incomplete poly(A) enrichment or abundant presence of immature transcripts.
- **Intergenic reads**: if a significant portion of reads is aligned outside of annotated gene sequences, this may suggest genomic DNA contamination (or abundant non-coding transcripts).
- **3' bias**: over-representation of 3' portions of transcripts indicates RNA degradation.

Read distribution For mRNA-seq, one would expect the majority of the aligned reads to overlap with exons. This assumption can be tested using the `read_distribution.py` script, which counts the numbers of reads overlapping with various gene- and transcript-associated genomic regions, such as exons and introns.

```
$ read_distribution.py -r ${REF_DIR}/sacCer3.bed \      # annotation file
    -i WT_1_Aligned.sortedByCoord.out.bam \ # runs only on single files

Total Reads                   7501551
Total Tags                    7565292
Total Assigned Tags           6977808
=====================================================================
Group               Total_bases         Tag_count           Tags/Kb
CDS_Exons           8832031             6970376             789.22
5'UTR_Exons         0                   0                   0.00
3'UTR_Exons         0                   0                   0.00
Introns             69259               6353                91.73
TSS_up_1kb          2421198             309                 0.13
TSS_up_5kb          3225862             309                 0.10
TSS_up_10kb         3377251             309                 0.09
TES_down_1kb        2073978             674                 0.32
TES_down_5kb        3185496             770                 0.24
TES_down_10kb       3386705             770                 0.23
=====================================================================
```

To compare the read distribution values for different samples, it is helpful to turn the text-based output of `read_distribution.py` into a bar graph:

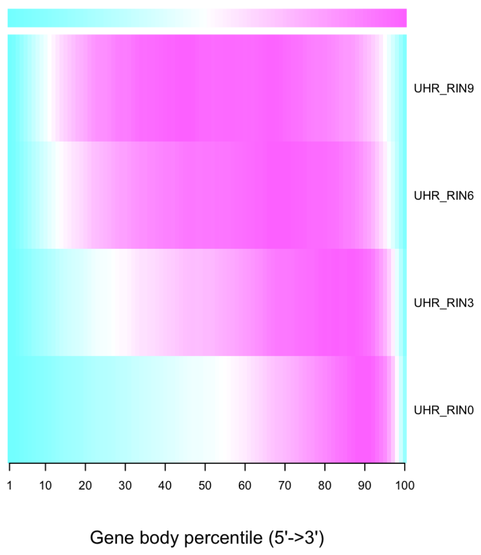


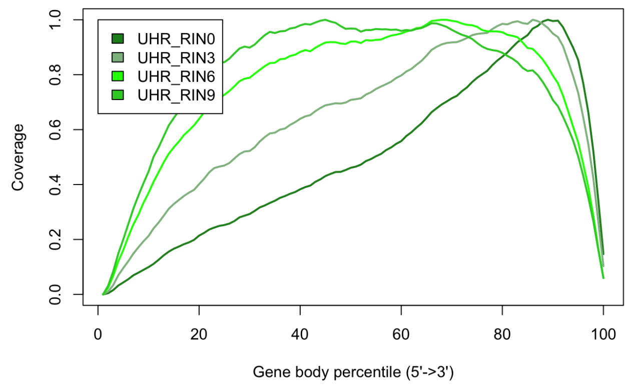
The .R script and `read_distribution.py` result files for this plot can be found at `https://github.com/friedue/course_RNA-seq2015`. You will probably want to change a couple of details, e.g. including intron counts. You can also let MultiQC do the work if you capture the output of `read_distribution.py` in a simple text file.

Gene body coverage To assess possible 3' or 5' biases, you can use `RSeQC`'s `geneBody_coverage.py` script. Given an annotation file with the transcript models of your choice, it will divide each transcript into 100 sections, count the reads overlapping with each section and generate two plots visualizing the general abundance of reads across all transcript bodies.

```
$ REF_DIR=~/mat/referenceGenomes/S_cerevisiae

# Generate an index for the BAM file
$ samtools index WT_1_Aligned.sortedByCoord.out.bam

$ geneBody_coverage.py \
  -i WT_1_Aligned.sortedByCoord.out.bam \ # aligned reads
  -r ${REF_DIR}/sacCer3.bed \ # annotation file
  -o geneBodyCoverage_WT_1 # output name

# if no plots are being generated automatically, the R script produced by the
    python script can be run manually:
$ <PATH to R installation>/bin/R < geneBodyCoverage_WT_1.geneBodyCoverage.r \
  --vanilla \ # tells R not to waste time trying to load previous sessions etc.
  --slave # makes R run in a less verbose mode
```

We ran the `geneBody_coverage.py` script on four human samples of RNA-seq with varying degrees of RNA quality, ranging from RIN = 0 (degraded) to RIN = 9 (high quality RNA) (see Section 1.1.1 for details about RIN). The resulting plots show varying degrees of 3' bias where samples with degraded RNA (RIN 0) show a more prominent bias than high-quality RNA (RIN 9).

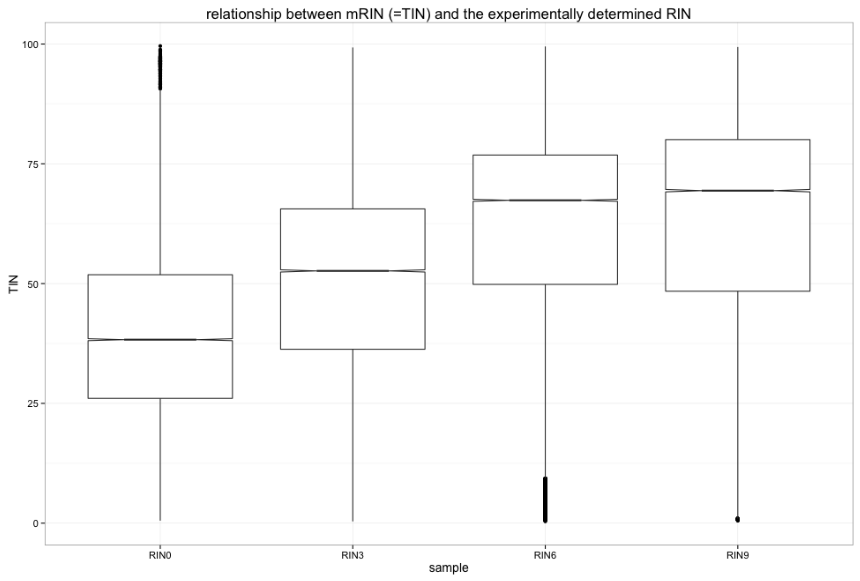




***in silico* mRIN calculation** The RNA integrity number (RIN, see Section 1.1.1) that is calculated during library preparation to assess the RNA quality is rarely indicated in the public data repositories. It might thus be informative to determine a measure of mRNA degradation *in silico*. `RSeQC`'s `tin.py` script does exactly that, using the deviation from an expected uniform read distribution across the gene body as a proxy (Feng et al., 2015).

```
$ tin.py -i WT_1_Aligned.sortedByCoord.out.bam -r ${REF_DIR}/sacCer3.bed
```

`tin.py` will generate a `.xls` file where the *in silico* mRIN is stored for each gene or transcript from the `BED` file. The second output file, `*summary.txt`, gives a quick overview of the mean and median values across all genes for a given sample. Using human samples with known, experimentally determined RIN numbers, we can see that the *in silico* mRIN does correlate:



You can find the `.R` script and `tin.py` result files underlying these box plots in the following github repository: `https://github.com/friedue/course_RNA-seq2015`.

3.4.3 Quality control with QoRTs

As an alternative to `RSeQC`, the Quality of RNA-Seq Toolset (`QoRTs`) was developed, which is a comprehensive and multifunctional toolset that assists in quality control and data processing of high-throughput RNA sequencing data. It creates many of the same output and plots as `RSeQC`, but the authors claim it is more accurate (Hartley and Mullikin, 2015).

The following command runs the complete `QoRTs` QC analysis (refer to Table 12 to see all the individual functions and commands).

```
$ REF_DIR=~/mat/referenceGenomes/S_cerevisiae

# to obtain the total number of raw reads you can make use of the calculator capabilities of bc
# (the number is an optional parameter for QoRTs though)
$ for FASTQ in ~/mat/precomputed/rawReads_yeast_Gierlinski/WT_1/ERR45849*gz; do zcat $FASTQ | wc -l ; done | paste -sd+ | bc | awk '{print $1/4}'

$ java -Xmx4g -jar ~/mat/software/qorts.jar QC  \
  --singleEnded  \ # QoRTs assumes the data is paired-end unless this flag is specified
  --seqReadCt 7014609 \ # total number of starting reads before mapping (see cmd above)
  --generatePdfReport WT_1_Aligned.sortedByCoord.out.bam \ # aligned reads
  ${REF_DIR}/sacCer3.gtf  \ # annotation file
  ./QoRTs_output/ # output folder
```

Note that by default, `QoRTs` assumes the data is paired end unless otherwise specified.

The run or exclude individual functions:

```
$ REF_DIR=~/mat/referenceGenomes/S_cerevisiae

# to only run a single function
$ java -Xmx4g -jar qorts.jar QC  \
  --singleEnded  \
   --runFunctions writeGeneBody \ # run only the genebody coverage function
  --generatePdfReport WT_1_Aligned.sortedByCoord.out.bam \
  ${REF_DIR}/sacCer3.gtf  \
  ./QoRTs_output/

# to exclude a function
$ java -Xmx4g -jar QoRTs.jar QC  \
  --singleEnded  \
   --skipFunctions  JunctionCalcs \ # run every function except the JunctionCalcs function
   --generatePdfReport WT_1_Aligned.sortedByCoord.out.bam \
  ${REF_DIR}/sacCer3.gtf  \
  ./QoRTs_output/
```

To include or exclude more than one function, use a comma-delimited list (without white spaces) of the respective functions.

An example QoRTs report can be found at `http://chagall.med.cornell.edu/RNASEQcourse/`.

3.4.4 Summarizing the results of different QC tools with `MultiQC`

In Section 2.3, we already made use of `MultiQC` (Ewels et al., 2016) for collapsing the results of `FastQC`, which we ran on every technical replicate. You can also generate a comprehensive report of the post-alignment QC using `MultiQC` as the tool can recognize the results of almost all the tools we discussed in this chapter:

- General post-alignment QC
  - the log files produced by `STAR`

  - `samtools flagstat`
  - results of `RSeQC`'s `bam_stat.py`

- RNA-seq-specific QC
  - read distribution (e.g., using `RSeQC` or `QoRTs`)
  - gene body coverage (e.g., using `RSeQC` or `QoRTs`)
  - the splice junction information obtained with `QoRTs`

```
# collect all QC results of interest in one folder, e.g. QC_collection
# subfolders can be be assigned for each sample, which will make the naming conventions used
# by MultiQC easier
# you can either copy or link the files that you need
$ ls QC_collection/WT_1/
geneBodyCoverage_WT_1.geneBodyCoverage.r          QC.NVC.minus.clipping.R1.txt.gz
geneBodyCoverage_WT_1.geneBodyCoverage.txt        QC.NVC.raw.R1.txt.gz
QC.b2nAZCenkhtb.log                               QC.NVC.tail.clip.R1.txt.gz
QC.biotypeCounts.txt.gz                           QC.QORTS_COMPLETED_OK
QC.chromCount.txt.gz                              QC.QORTS_COMPLETED_WARN
QC.cigarOpDistribution.byReadCycle.R1.txt.gz      QC.quals.r1.txt.gz
QC.cigarOpLengths.byOp.R1.txt.gz                  QC.r3z9iUXrtnHr.log
QC.exonCounts.formatted.for.DEXSeq.txt.gz         QC.spliceJunctionAndExonCounts.forJunctionSeq.txt.gz
QC.fxNgbBmcKJnC.log                               QC.spliceJunctionCounts.knownSplices.txt.gz
QC.gc.byRead.txt.gz                               QC.spliceJunctionCounts.novelSplices.txt.gz
QC.gc.byRead.vsBaseCt.txt.gz                      QC.summary.txt
QC.geneBodyCoverage.byExpr.avgPct.txt.gz          read_distribution.txt
QC.geneBodyCoverage.by.expression.level.txt.gz    rseqc_bam_stat.txt
QC.geneBodyCoverage.DEBUG.intervals.txt.gz        samtools_flagstat.txt
QC.geneBodyCoverage.genewise.txt.gz               WT_1Log.final.out
QC.geneCounts.formatted.for.DESeq.txt.gz          WT_1Log.out
QC.geneCounts.txt.gz                              WT_1Log.progress.out
QC.NVC.lead.clip.R1.txt.gz

# QoRTs results
$ ls QC_collection/WT_1/QC*
# STAR Log files
$ ls QC_collection/WT_1/*Log*out

# the folder also contains (somewhat arbitrarily named) results of individual RSeQC scripts
# including bam_stat.py, read_distribution.py, geneBody_coverage.py

# run MultiQC
$ cd QC_collection/
$ ~/mat/software/anaconda2/bin/multiqc . \
    --dirs \ # use the names of the subdirectories
    --ignore ERR*  \ # ignoring FastQC results in case they are there
    --filename multiQC_align
```

4 Read Quantification

4.1 Gene-based read counting

To compare the expression of single genes between different conditions, an essential step is the quantification of reads per gene. In principle, the counting of reads overlapping with genomic features is a fairly simple task, but there are some details that need to be decided on depending on the nature of your experiment and the desired outcome (Figure 12).

When counting reads, make sure you know how the program handles the following:

- overlap size (full read vs. partial overlap)
- multi-mapping reads
- reads overlapping multiple genomic features of the same kind
- reads overlapping introns

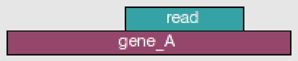
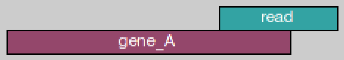
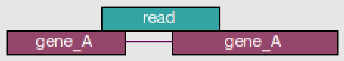
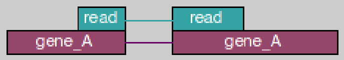
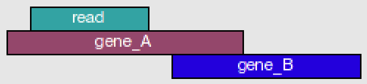
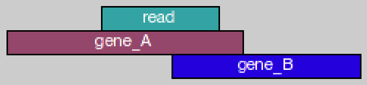
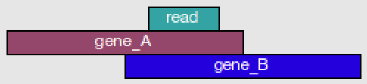
	union	intersection _strict	intersection _nonempty
	gene_A	gene_A	gene_A
	gene_A	no_feature	gene_A
	gene_A	no_feature	gene_A
	gene_A	gene_A	gene_A
	gene_A	gene_A	gene_A
	ambiguous	gene_A	gene_A
	ambiguous	ambiguous	ambiguous

Figure 12: The `htseq-count` script of the `HTSeq` suite offers three different modes to handle details of read–feature overlaps that are depicted here. The default of `featureCounts` is the behavior of the `union` option. Image taken from `http://www-huber.embl.de/users/anders/HTSeq/doc/count.html`.

The most popular tools for gene quantification are `htseq-count` and `featureCounts`. Both are part of larger tool packages (Anders et al., 2014; Liao et al., 2014). `htseq-count` offers three different modes to tune its behavior to define overlap instances (Figure 12). The recommended mode is `union`, which counts overlaps even if a read only shares parts of its sequence with a genomic feature and disregards reads that overlap more than one feature. This is similar to `featureCounts` that calls a hit if any overlap (1 bp or more) is found between the read and a feature and provides the option to either exclude multi-overlap reads or to count them for each feature that is overlapped.

In addition to the nature and lengths of the reads, gene expression quantification will be strongly affected by the underlying gene models that are usually supplied to the quantification programs via GTF or BED(-like) files (see Section 3.1 for details on the file formats and annotations).

The following commands will count the number of reads overlapping with genes using `featureCounts`.

```
# count reads per gene
$ ~/mat/software/subread-1.6.0-Linux-x86_64/bin/featureCounts  \
    -a ${REF_DIR}/sacCer3.gtf \
    -o featureCounts_results.txt \
    alignment/*bam # use all BAM files in the folder "alignment"
```

The output of `featureCounts` consists of two files:

1. The one defined by the `-o` parameter (e.g., `featureCounts_results.txt`) – this one contains the actual read counts per gene (with gene ID, genomic coordinates of the gene including strand and length); the first line (starting with #) contains the command that was used to generate the file.
2. A file with the suffix `.summary`: This file gives a quick overview about how many reads could be assigned to genes and the reasons why some of the could not be assigned. This is a very useful file to double check the settings you've chosen for the counting.

`featureCounts` also allows to count reads overlapping with individual exons.

```
# count reads per exon
$ ~/mat/software/subread-1.6.0-Linux-x86_64/bin/featureCounts  \
    -a ${REF_DIR}/sacCer3.gtf \
    -f \ # count read overlaps on the feature level
    -t exon \ # feature type
    -O \ # allow reads to overlap more than one exon
    -o featCounts_exons.txt \
    alignment/*bam
```

However, there are (at least) two caveats here:

- If an exon is part of more than one isoform in the annotation file, `featureCounts` will return the read counts for the same exon multiple times ($n = number\ of\ transcripts\ with\ that\ exon$). Make sure you remove those multiple entries in the result file before the differential expression anaysis, e.g., using a UNIX command[*] or within R.
- If you want to assess differential expression of exons, it is highly recommended to create an annotation file where overlapping exons of different isoforms are split into artificially disjoint bins before applying `featureCounts`. See, for example, Anders et al. (2012). To create such a "flattened" annotation file from a GTF file (Section 3.1.1), you can use the `dexseq_prepare_annotation.py` script of the DEXSeq package (Anders et al., 2012) and the section "Preparing the annotation" of the corresponding vignette at bioconductor. Alternatively, you can use QoRTs to prepare the proper annotation, too[†].

4.2 Isoform counting methods

The previously discussed methods count the number of fragments that can be assigned to a gene as a whole. There is another school of thought that insists that quantifying reads that originated from transcripts should also be done on the transcript level[‡]. So far, most comparisons of methods point towards superior results of gene-based quantification (which is why we adhere to it for now) and there is no standard technique for summarizing expression levels of genes with several isoforms (see, for example, Soneson et al. (2015), Dapas et al. (2016), Germain et al. (2016), and (Teng et al., 2016) for detailed comparisons of transcript-level quantifications).

[*] e.g., `sort -k2,2n -k3,3n featureCounts_exons.txt | uniq`

[†] see `https://hpc.nih.gov/apps/QoRTs/example-walkthrough.pdf` for details

[‡] For arguments in favor of the transcript-focused school of thought, see, e.g., Trapnell et al. (2013) and Pimentel's talk.

One trend seems to be clear though: the simple count-based approaches tend to underperform when they are used to determine transcript-level counts because they generally tend to ignore reads that overlap with more than one feature. While this is reasonable when the features are entire genes, this leads to an enormous number of discarded reads when quantifying different isoforms since multiple transcripts of the same gene naturally tend to overlap.

In order to quantify isoforms, you should perhaps look into different programs, e.g., Cufflinks (Trapnell et al., 2012), RSEM (Li and Dewey, 2011), eXpress (Roberts and Pachter, 2013) – these tools have been around the longest and are therefore most often cited. These tools typically use a *deBruijn* graph approach to assign reads to a given isoform if they are compatible with that transcript structure (see Figure 13 for a simple example).

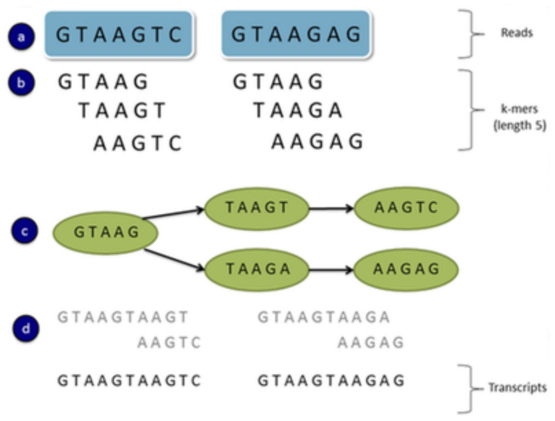


Figure 13: Schema of a simple *deBruijn* graph-based transcript assembly. (A) Read sequences are split into (B) all subsequence k-mers (here: of length 5) from the reads. (C) A *deBruijn* graph is constructed using unique k-mers as the nodes and overlapping k-mers connected by edges (a k-mer shifted by one base overlaps another k-mer by $k-1$ bases). (D) The transcripts are assembled by traversing the two paths in the graph. Figure taken from Moreton et al. (2015)

Very recently, two groups published algorithms that are based on the idea that it may not be important to exactly know *where* within a transcript a certain read originated from. Instead, it may be enough to simply know *which* transcript the read represents. These algorithms therefore do not generate a `BAM` file because they do not worry about finding the best possible alignment. Instead, they yield a (probabilistic) measure of how many reads indicate the presence of each transcript. The tools are:

- Sailfish (Patro et al., 2014), or the more updated, more accurate version, Salmon (Patro et al., 2017)
- kallisto (Bray et al., 2016)

While these approaches are extremely fast compared to the usual alignment–counting routines that we have described at length, they cannot be used to detect novel isoforms.

Instead of direct isoform quantification, you may be able to glean more accurate answers from alternative approaches, e.g., quantification of exons (Anders et al., 2012)[§] or estimates of alternative splicing events such as exon skipping, intron retention etc. (e.g., MISO (Katz et al., 2010), rMATS (Shen et al., 2014)).

The main take home message here is once again: Know your data and your question, and research the individual strengths and pitfalls of the individual tools before deciding which one to use. For example, one major issue reported for Cufflinks is its inability to handle single-exon transcripts. Therefore, you should avoid using it if you are dealing with a fairly simple transcriptome (Kanitz et al., 2015). On the other hand, transcriptome reconstruction as attempted by Cufflinks generates large amounts of false positives (as well as false negatives) in very complicated transcriptomes, such as the human one while it seems to hit a better spot when applied to moderately complex transcriptomes such as the one of *C. elegans* (Jänes et al., 2015). In comparison, the novel lightweight quantification algorithms perform well for isoform quantification of known transcriptomes, but they are naturally very sensitive to incomplete or changing annotation. In addition, it is not entirely clear yet whether the resulting values can be used with the established algorithms to determine differential gene expression (Soneson et al., 2015; Pimentel et al., 2016).

[§]The above shown `featureCounts`-based exon counting should not be used with `DEXSeq` unless exons with varying boundaries have been divided into disjoint bins (Anders et al., 2012; Teng et al., 2016; Soneson et al., 2016).

© 2015-2018 Applied Bioinformatics Core | Weill Cornell Medical College

! The main caveats of assigning reads to transcripts are:

- inconsistent annotation of transcripts
- multiple isoforms of widely differing lengths
- anti-sense/overlapping transcripts of different genes

There is no really good solution yet! Be careful with your conclusions and if possible, limit your analyses to gene-based approaches.

5 Normalizing and Transforming Read Counts

Given a uniform sampling of a diverse transcript pool, the number of sequenced reads mapped to a gene depends on:

- its own expression level,
- its length,
- the sequencing depth,
- the expression of all other genes within the sample.

In order to compare the gene expression *between two conditions*, we must therefore calculate the fraction of the reads assigned to each gene relative to the total number of reads and with respect to the entire RNA repertoire which may vary drastically from sample to sample. While the number of sequenced reads is known, the total RNA library and its complexity is unknown and variation between samples may be due to contamination as well as biological reasons. The purpose of normalization is to eliminate systematic effects that are not associated with the biological differences of interest. See Table 13 for details on the most commonly used normalization methods that deal with these issues in different ways.

5.1 Normalization for sequencing depth differences

As shown in Figure 14, the size factor method implemented by the R package `DESeq` leads to relatively similar read count distribution between different libraries. We will now use the output of `featureCounts` (= raw read counts), read them into R and normalize the read counts for sequencing depth differences with `DESeq`.

5.1.1 `DESeq`'s specialized data set object

`DESeq` stores virtually all information associated with your experiment in one specific R object, called `DESeqDataSet`. This is, in fact, a specialized object of the class "SummarizedExperiment". This, in turn, is a container where rows (`rowRanges()`) represent features of interest (e.g. genes, transcripts, exons) and columns represent samples (`colData()`). The actual count data is stored in the `assay()` slot. More specifically:

- `colData` is a `data.frame` that can contain all the variables you know about your samples, such as the experimental condition, the type and date of sequencing and so on (see Section 1.4). Its `row.names` should correspond to the *unique* sample names.
- `rowRanges` is meant to keep all the information about the genes (which are genomic ranges, i.e., defined by chromosome, start, end, strand and ID)
- `assay` should contain a matrix of the actual values associated with the genes and samples. For `DESeq`, this is overlapping with `countData`.

We will first read in the read counts, which will eventually be stored in `countData`.

```
### Open an R console, e.g. using RStudio
# code lines starting with `>` indicate the R console
> library(magrittr) # this will allow us to string commands together in a UNIX-
    pipe-like fashion using %>%

# get the table of read counts
> read.counts <- read.table("featureCounts_result.txt", header = TRUE)

# the gene IDs should be stored as row.names
> row.names(readcounts) <- readcounts$Geneid

# exclude all columns that do not contain read counts
> readcounts <- readcounts[ , -c(1:6)]

```

```
# give meaningful sample names - this can be achieved via numerous approaches
# the one shown here is the least generic and most error-prone one!
> orig_names <- names(readcounts)
> names(readcounts) <- c("SNF2_1", "SNF2_2", "SNF2_3", "SNF2_4", "SNF2_5", "WT_
  1", "WT_2", "WT_3", "WT_4", "WT_5")

# alternative way to assign the sample names, which reduces the
# potential for typos as well as for the wrong order:
> names(readcounts) <- gsub(".*(WT|SNF2)(_[0-9]+).*", "\\1\\2", orig_names)

# ALWAYS CHECK YOUR DATA AFTER YOU MANIPULATE IT!
> str(readcounts)
> head(readcounts)
          SNF2_1 SNF2_2 SNF2_3 SNF2_4 SNF2_5 WT_1 WT_2 WT_3 WT_4 WT_5
YAL012W     7347   7170   7643   8111   5943 4309 3769 3034 5601 4164
YAL069W        0      0      0      0      0    0    0    0    0    0
YAL068W-A      0      0      0      0      0    0    0    0    0    0
YAL068C        2      2      2      1      0    0    0    0    2    2
YAL067W-A      0      0      0      0      0    0    0    0    0    0
YAL067C      103     51     44     90     53   12   23   21   30   29
```

Now that we have the read counts, we also need some information about the samples, which will be stored in `colData`. As described above, this should be a `data.frame`, where the rows match the column names of the count data we just generated. In addition, each row should contain information about the condition of each sample (here: WT and SNF2 [knock-out]).

```
# make a data.frame with meta-data where row.names should match the individual
# sample names
> sample_info <- data.frame(condition = gsub( "_[0-9]+", "", names(readcounts)
  ),
                            row.names = names(readcounts) )
> sample_info
       condition
SNF2_1      SNF2
SNF2_2      SNF2
SNF2_3      SNF2
SNF2_4      SNF2
SNF2_5      SNF2
WT_1          WT
WT_2          WT
WT_3          WT
WT_4          WT
WT_5          WT

# IF NEEDED, install DESeq2, which is not available via install.packages(),
# but through bioconductor
> source("http://bioconductor.org/biocLite.R")
> biocLite("DESeq2")
> library(DESeq2)

# generate the DESeqDataSet
> DESeq.ds <- DESeqDataSetFromMatrix(countData = readcounts,
                                     colData = sample_info,
                                     design = ~ condition)

# you can check the result using the accessors described above:
> colData(DESeq.ds) %>% head
> assay(DESeq.ds) %>% head
> rowRanges(DESeq.ds) %>% head
```

In addition to the general functions and accessors that work for "SummarizedExperiment" objects*, the creators of DESeq2 have added more functions to their specific object, for example `counts()`.

```
# test what counts() returns
> counts(DESeq.ds) %>% str

# remove genes without any counts
> DESeq.ds <- DESeq.ds[ rowSums(counts(DESeq.ds)) > 0, ]

# investigate different library sizes
> colSums(counts(DESeq.ds)) # should be the same as colSums(readcounts)
```

`DESeq's` default method to normalize read counts to account for differences in sequencing depths is implemented in `estimateSizeFactors` (see Table 13).

```
# calculate the size factor and add it to the data set
> DESeq.ds <- estimateSizeFactors(DESeq.ds)
> sizeFactors(DESeq.ds)

# counts() allows you to immediately retrieve the _normalized_ read counts
> counts.sf_normalized <- counts(DESeq.ds, normalized = TRUE)
```

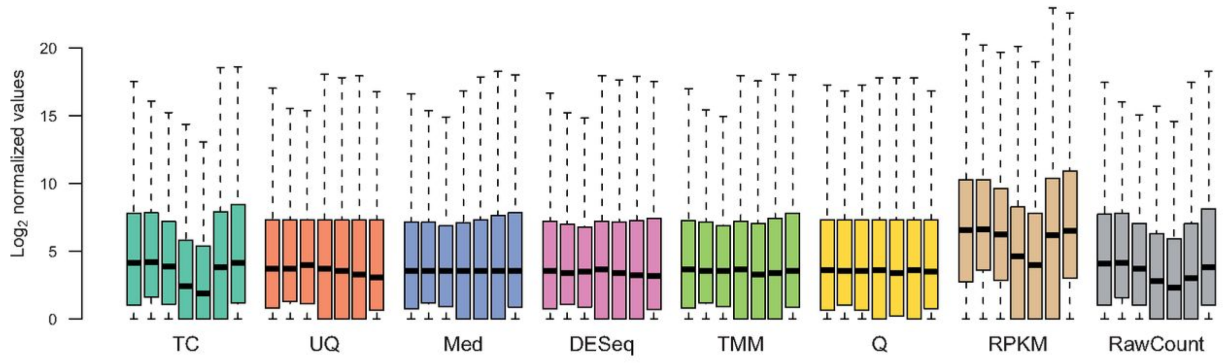


Figure 14: Figure from Dillies et al. (2013) that shows the effects of different approaches to normalize for read count differences due to library sizes (TC, total count; UQ, upper quartile; Med, median; DESeq, size factor; TMM, Trimmed Mean of M-values; Q, quantile) or gene lengths (RPKM). See Tables 13 and 14 for details of the different normalization methods.

While the majority of normalization methods work well, **RPKM** and **total count** normalization should be **avoided** in the context of DE analysis, no matter how often you see them applied in published studies. RPKM, FPKM etc. are only needed if expression values need to be compared *between different genes* within the *same sample* for which the different gene lengths must be taken into consideration.

1. Name two technical reasons why the read count for the same gene may vary *between two samples* although it is not differentially expressed.

2. Name two technical reasons why the read counts of two genes may vary *within the same sample* although they are expressed at the same level.

*`https://www.rdocumentation.org/packages/SummarizedExperiment/versions/1.2.3/topics/RangedSummarizedExperiment-class`

© 2015-2018 Applied Bioinformatics Core | Weill Cornell Medical College

5.2 Transformation of sequencing-depth-normalized read counts

While most models used for differential gene expression testing operate on the raw count values, many downstream analyses (including clustering) work better if the read counts are *transformed* to the *log* scale. While you will occasionally see log_{10} transformed read counts, log_2 is more commonly used because it is easier to think about doubled values rather than powers of 10. The transformation should be done *in addition* to sequencing depth normalization.

5.2.1 Log_2 transformation of read counts

```
# transform size-factor normalized read counts to log2 scale using a
  pseudocount of 1
> log.norm.counts <- log2(counts.sf_normalized + 1)
```

You can see how the log_2 transformation makes even simple graphs more easily interpretable by generating boxplots of read counts similar to the ones in Figure 14:

```
> par(mfrow=c(2,1)) # to plot the following two images underneath each other

# first, boxplots of non-transformed read counts (one per sample)
> boxplot(counts.sf_normalized, notch = TRUE,
          main = "untransformed read counts", ylab = "read counts")

# box plots of log2-transformed read counts
> boxplot(log.norm.counts, notch = TRUE,
          main = "log2-transformed read counts",
          ylab = "log2(read counts)")
```

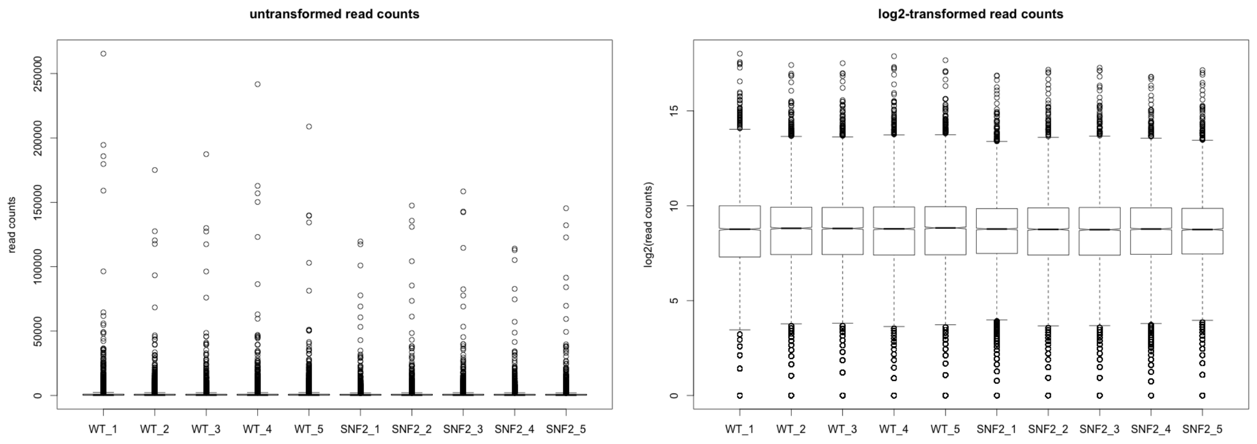


Figure 15: Comparison of the read distribution plots for untransformed and log_2-transformed values.

5.2.2 Visually exploring normalized read counts

To get an impression of how similar read counts are between replicates, it is often insightful to simply plot the counts in a pairwise manner (Figure 16, upper panels). This can be achieved with the basic, but versatile `plot()` function:

```
plot(log.norm.counts[,1:2], cex=.1, main = "Normalized log2(read counts)")
```

Many statistical tests and analyses assume that data is homoskedastic, i.e. that all variables have similar variance. However, data with large differences among the sizes of the individual observations often shows heteroskedastic behavior. One way to visually check for heteroskedasticity is to plot the mean vs. the standard deviation (Figure 16, lower panel).

```
# IF NEEDED, install the vsn package
> source("http://bioconductor.org/biocLite.R")
> biocLite("vsn")

# mean-sd plot
> library(vsn)
> library(ggplot2)
> msd_plot <- meanSdPlot(log.norm.counts,
                         ranks=FALSE, # show the data on the original scale
                         plot = FALSE)
> msd_plot$gg +
      ggtitle("sequencing depth normalized log2(read counts)") +
      ylab("standard deviation")
```

The y-axis shows the variance of the read counts across all samples. Some variability is, in fact, expected, but the clear hump on the left-hand side indicates that for read counts < 32 ($2^5 = 32$), the variance is higher than for those with greater read counts. That means that there is a dependence of the variance on the mean, which violates the assumption of homoskedasticity.

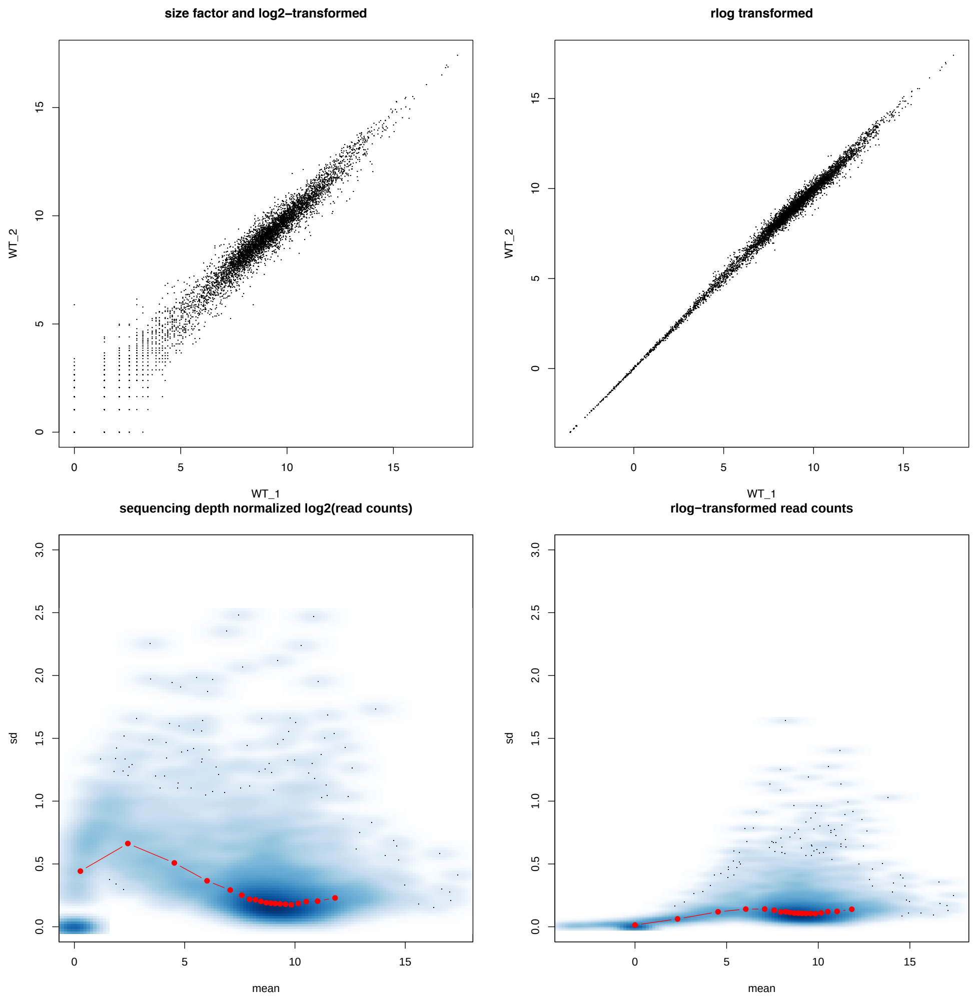


Figure 16: Comparison of log_2- and $rlog$-transformed read counts. The upper panel shows simple pairwise comparisons of replicate samples; the lower panel contains mean-sd-plots based on all samples of the experiment.

5.2.3 Transformation of read counts including variance shrinkage

To reduce the amount of heteroskedasticity, `DESeq2` and also `edgeR` offer several means to shrink the variance of low read counts. They do this by using the dispersion-mean trend that can be observed for the entire data set as a reference. Consequently, genes with low and highly variable read counts will be assigned more homogeneous read count estimates so that their variance resembles the variance observed for the majority of the genes (which hopefully have a more stable variance).

`DESeq2`'s `rlog()` function returns values that are both normalized for sequencing depth and transformed to the log_2 scale where the values are adjusted to fit the experiment-wide trend of the variance-mean relationship.

```
# obtain regularized log-transformed values
> DESeq.rlog <- rlog(DESeq.ds, blind = TRUE)
> rlog.norm.counts <- assay(DESeq.rlog)

# mean-sd plot for rlog-transformed data
> library(vsn)
> library(ggplot2)
> msd_plot <- meanSdPlot(rlog.norm.counts,
                         ranks=FALSE, # show the data on the original scale
                         plot = FALSE)
> msd_plot$gg +
      ggtitle("rlog-transformed read counts") +
      ylab("standard deviation")
```

The `rlog()` function's `blind` parameter should be set to `FALSE` if the different conditions lead to strong differences in a large proportion of the genes. If `rlog()` is applied without incorporating the knowledge of the experimental design (`blind = TRUE`, the default setting), the dispersion will be greatly overestimated in such cases.

5.3 Exploring global read count patterns

An important step before diving into the identification of differentially expressed genes is to check whether expectations about basic global patterns are met. For example, technical and biological replicates should show similar expression patterns while the expression patterns of, say, two experimental conditions should be more dissimilar. There are multiple ways to assess the similarity of expression patterns, we will cover the three that are used most often for RNA-seq data.

5.3.1 Pairwise correlation

The *Pearson correlation coefficient*, r, is a measure of the strength of the linear relationship between two variables and is often used to assess the similarity of RNA-seq samples in a pair-wise fashion. It is defined as the covariance of two variables divided by the product of their standard deviation. The ENCODE consortium recommends that "for messenger RNA, (...) biological replicates [should] display $>$0.9 correlation for transcripts/features".

In R, pairwise correlations can be calculated with the `cor()` function.

5.3.2 Hierarchical clustering

To determine whether the different sample types can be separated in an unsupervised fashion (i.e., samples of different conditions are more dissimilar to each other than replicates within the same condition), hierarchical clustering can be used. Hierarchical clustering is typically based on pairwise comparisons of individual samples, which are grouped into "neighborhoods" of similar samples. The basis of hierarchical clustering is therefore a matrix of similarity metrics (which is different from the actual gene expression values!).

Hierarchical clustering requires two decisions:

1. How should the (dis)similarity between pairs be calculated?
2. How should the (dis)similarity be used for the clustering?

A common way to assess the (dis)similarity is the Pearson correlation coefficient, r, that we just described. The corresponding *distance measure* is $d = 1 - r$. Alternatively, the Euclidean distance is often used as a measure of distance between two vectors of read counts. The Euclidean distance is strongly influenced by differences of the scale: if two samples show large differences in sequencing depth, this will affect the Euclidean distance more than the distance based on the Pearson correlation coefficient.

Just like there are numerous ways to calculate the distance, there are multiple options to decide on how the distances should be used to define clusters of samples. The most popular choices for the *linkage function* are

Table 6: Comparison of unsupervised classification and clustering techniques. The following table was adapted from Karimpour-Fard et al. (2015); see that publication for more details on additional (supervised) classification methods such as support vector machines. Classifiers try to reduce the number of features that represent the most prevalent patterns within the data. Clustering techniques aim to group similar features.

	Method	What does it do?	How?	Strengths	Weaknesses	Sample size
Classification	PCA	Separates features into groups based on commonality and reports the weight of each component's contribution to the separation	Orthogonal transformation; transfers a set of correlated variables into a new set of uncorrelated variables	Unsupervised, nonparametric, useful for reducing dimensions before using supervision	Number of features must exceed number of treatment groups	Number of features must exceed number of treatment groups
	ICA	Separates features into groups by eliminating correlation and reports the weight of each componentâĂŹs contribution to the separation	Nonlinear, non-orthogonal transformation; standardizes each variable to a unit variance and zero mean	Works well when other approaches do not because data are not normally distributed	Features are assumed to be independent when they actually may be dependent	Unlimited sample size; data non-normally distributed
Clustering	K-means	Separates features into clusters of similar expression patterns	Compares and groups magnitudes of changes in the means into K clusters where K is defined by the user	Easily visualized and intuitive; greatly reduces complexity; performs well when distance information between data points is important to clustering	Sensitive to initial conditions and user-specified number of clusters (K)	Best with a limited dataset, i.e., ca. 20 to 300 features
	Hierarchical	Clusters treatment groups, features, or samples into a dendrogram	Compares all samples using either agglomerative or divisive algorithms with distance and linkage functions	Unsupervised; easily visualized and intuitive	Does not provide feature contributions; not iterative, thus sensitive to cluster distance measures and noise and outliers	Best with a limited dataset, i.e., ca. 20 to 300 features or samples

- *complete*: intercluster distance ≡ largest distance between any 2 members of either cluster
- *average*: intercluster distance ≡ average distance between any 2 members
- *single*: intercluster distance ≡ shortest distance between any 2 members

! Avoid “single" linkage on gene expression data; “complete" and “average" linkage tend to be much more appropriate, with “complete" linkage often outperforming “average" (Gibbons and Roth, 2002).

The result of hierarchical clustering is a *dendrogram* (Figure 17); clusters are obtained by cutting the dendrogram at a level where the jump between two consecutive nodes is large: connected components then form individual clusters. It must be noted that there is no consensus on how to decide the “correct" number of clusters. The cluster structure recovered by the *in silico* clustering does not necessarily represent the “true" structure of the data[†]. As Yona et al. (2009) point out: “The application of any clustering algorithm will result in some partitioning of the data into groups, (...) the choice of the clustering algorithm may greatly affect the outcome (...) and their output may vary a great deal, depending on the starting point." While statistical approaches to validating cluster choices exist[‡], for most applications in RNA-seq analyses it will

[†]The true structure of the data will, in turn, be affected by and reflect the effect of random noise, technical artefacts and biological variability.

[‡]See, for example, the vignette to the R package `clValid`: `https://cran.r-project.org/web/packages/clValid/vignettes/clValid.pdf`.

suffice to judge the clustering results based on your prior knowledge of the experiment. In addition, the structure of the dendrogram should yield compact, well-defined clusters.

A dendrogram can be generated in R using the functions `cor()`, `as.dist()`, and `hclust()`:

```
# cor() calculates the correlation between columns of a matrix
> distance.m_rlog <- as.dist(1 - cor(rlog.norm.counts, method = "pearson" ))

# plot() can directly interpret the output of hclust()
> plot( hclust(distance.m_rlog),
  labels = colnames(rlog.norm.counts),
  main = "rlog transformed read counts\ndistance: Pearson correlation")
```

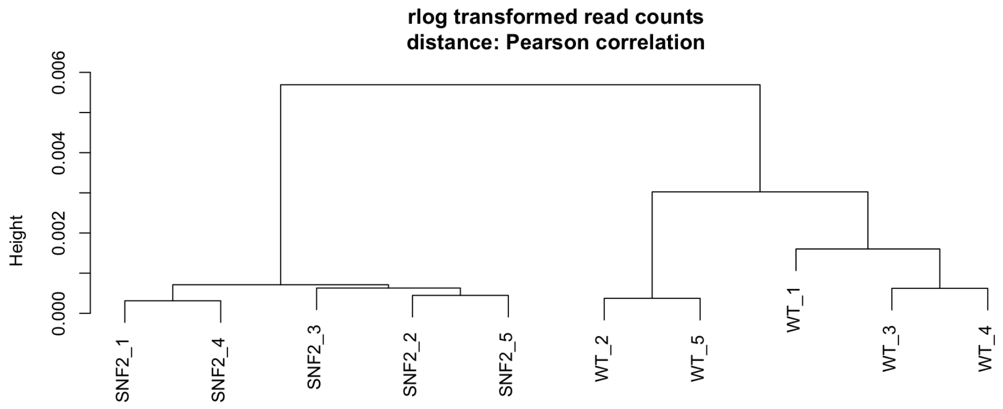


Figure 17: Dendrogram of *rlog*-transformed read counts for ten different samples, using the "complete" linkage function. The two conditions, SNF2 and WT, are well separated.

1. Which linkage method was used for the dendrogram generated with the code shown above?
2. Can you make a dendrogram with Euclidean distance and linkage method "average"?

5.3.3 Principal Components Analysis (PCA)

A complementary approach to determine whether samples display greater variability between experimental conditions than between replicates of the same treatment is principal components analysis. It is a typical example of dimensionality reduction approaches that have become very popular in the field of machine learning. The goal is to find *groups* of features (e.g., genes) that have something in common (e.g., certain patterns of expression across different samples), so that the information from thousands of features is captured and represented by a reduced number of groups.

The result of PCA are principal components that represent the directions along which the variation in the original multi-dimensional data matrix is maximal. This way a few dimensions (components) can be used to represent the information from thousands of mRNAs. This allows us to, for example, visually represent the variation of the gene expression for different samples by using just the top two PCs[§] as coordinates in a simple xy plot (instead of plotting thousands of genes per sample). Most commonly, the two principal components explaining the majority of the variability are displayed. It is also useful to identify unexpected patterns, such as batch effects or outliers. But keep in mind that PCA is not designed to discover unknown groupings; it is up to the researcher to actually identify the experimental or technical reason underlying the principal components. For more technical details and PCA alternatives depending on the types of data that you have, see, for example, Meng et al. (2016).

PCA can be performed in base R using the function `prcomp()`.

```
> pc <- prcomp(t(rlog.norm.counts))
> plot(pc$x[,1], pc$x[,2],
      col = colData(DESeq.ds)[,1],
```

[§]Per definition, PCs are ordered by reducing variability, i.e. the first PC will always be the component that captures the most variability.

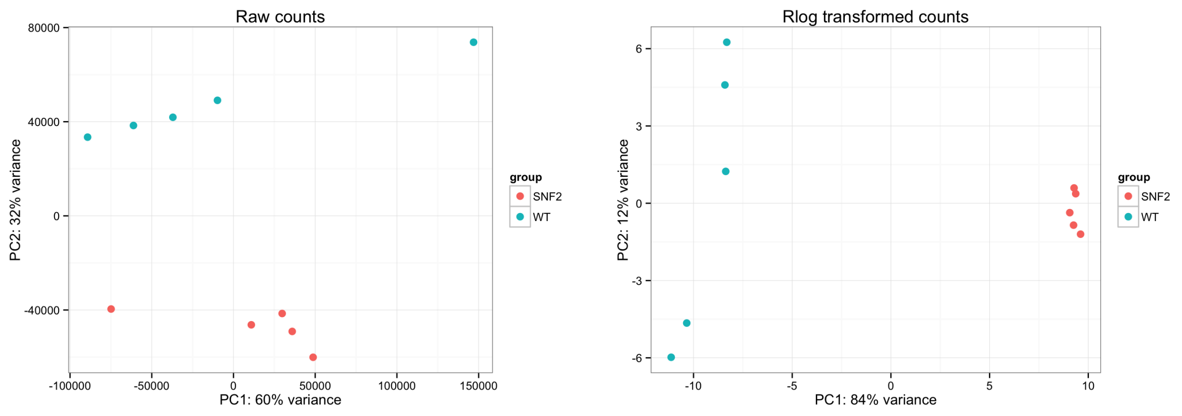


Figure 18: PCA on raw counts and *rlog*-transformed read counts with the DESEq2 convenience function plotPCA(). As indicated by the labels of the axes, the different sample types explain a greater fraction of the variance for *rlog*-transformed values than for the raw counts.

```
    main = "PCA of seq.depth normalized\n and rlog-transformed read counts")
```

DESeq2 also offers a convenience function based on ggplot2 to do PCA directly on a DESeqDataSet:

```
> library(DESeq2)
> library(ggplot2)

# PCA
> P <- plotPCA(DESeq.rlog)

# plot cosmetics
> P <- P + theme_bw() + ggtitle("Rlog transformed counts")
> print(P)
```

! PCA and clustering should be done on normalized and preferably transformed read counts, so that the high variability of low read counts does not occlude potentially informative data trends.

6 Differential Gene Expression Analysis

The two basic tasks of all DGE tools are:

1. Estimate the *magnitude* of differential expression between two or more conditions based on read counts from replicated samples, i.e., calculate the fold change of read counts, taking into account the differences in sequencing depth and variability.
2. Estimate the *significance* of the difference and correct for multiple testing.

The best performing tools tend to be `edgeR` (Robinson et al., 2010), `DESeq/DESeq2` (Anders and Huber, 2010; Love et al., 2014), and `limma-voom` (Ritchie et al., 2015) (see Rapaport et al. (2013); Soneson and Delorenzi (2013); Schurch et al. (2015) for reviews of DGE tools). `DESeq` and `limma-voom` tend to be more conservative than `edgeR` (better control of false positives), but `edgeR` is recommended for experiments with fewer than 12 replicates (Schurch et al., 2015). These tools are all based on the `R` language and make heavy use of numerous statistical methods that have been developed and implemented over the past two decades. They all follow the same approach, i.e., they estimate the gene expression difference for a given gene using regression-based models, followed by a statistical test based on the null hypothesis that the difference is close to zero (which would mean that there is no difference in the gene expression values that could be explained by the conditions). See Table 7 for a summary of the key properties of the most popular DGE tools; the next two sections will explain some more details of the two key steps of the DGE analyses.

Table 7: Comparison of programs for differential gene expression identification. Information shown here is based on the user guides of `DESeq2`, `edgeR`, `limmaVoom` and Rapaport et al. (2013), Seyednasrollah et al. (2015), and Schurch et al. (2015). LRT stands for log-likelihood ratio test.

Feature	**DESeq2**	**edgeR**	**limmaVoom**	**Cuffdiff**
Seq. depth normalization	Sample-wise size factor	Gene-wise trimmed median of means (TMM)	Gene-wise trimmed median of means (TMM)	FPKM-like or DESeq-like
Dispersion estimate	Cox-Reid approximate conditional inference with focus on maximum *individual* dispersion estimate	Cox-Reid approximate conditional inference moderated towards the *mean*	squeezes gene-wise residual variances towards the global variance	
Assumed distribution	Neg. binomial	Neg. binomial	*log*-normal	Neg. binomial
Test for DE	Wald test (2 factors); LRT for multiple factors	exact test for 2 factors; LRT for multiple factors	*t*-test	*t*-test
False positives	Low	Low	Low	High
Detection of differential isoforms	No	No	No	Yes
Support for multi-factored experiments	Yes	Yes	Yes	No
Runtime (3-5 replicates)	Seconds to minutes	Seconds to minutes	Seconds to minutes	Hours

! All statistical methods developed for read counts rely on approximations of various kinds, so that assumptions must be made about the data properties. `edgeR` and `DESeq`, for example, assume that the majority of the transcriptome is *unchanged* between the two conditions. If this assumption is not met by the data, both log_2 fold change and the significance indicators are most likely incorrect!

6.1 Estimating the difference between read counts for a given gene

To determine whether the read count differences between different conditions for a given gene are greater than expected by chance, DGE tools must find a way to estimate that difference. `edgeR` (Robinson et al., 2010), `DESeq/DESeq2` (Anders and Huber, 2010; Love et al., 2014), and `limma-voom` (Ritchie et al., 2015) all use regression models that are applied to every single gene. Linear regression models usually take the following typical form: $Y = b_0 + b_1 * x + e$

Here, Y will entail *all* read counts (from all conditions) for a given gene. b_0 is called the *intercept*; x is the condition (for RNA-seq, this is very often a discrete factor, e.g., "WT" or "mutant", or, in mathematical terms, 0 or 1), e is a term capturing the error or uncertainty, and b_1 is the coefficient that captures the difference. You may have encountered linear models in different contexts, they are usually used to predict unknown values of Y, i.e., one often wants to find a function that returns Y at any given point along a certain trajectory captured by the model. In the case of RNA-seq, we are actually more interested in the values that the coefficients take, more precisely, the parameter that explains the difference between groups of expression values belonging to different sets of x.

The very simple model shown above could be fitted in `R` using the function `lm(rlog.norm~genotype)`*, which will return estimates for both b_0 and b_1, so that the average expression values of the baseline genotype (e.g., WT = 0) would correspond to $Y = b_0 + b_1 * 0 + e$. This is equivalent to $Y = b_0$ (assuming that e is very small), thereby demonstrating why the intercept (b_0) can be interpreted as the average of our baseline group. b_1, on the other hand, will be the coefficient whose closeness to zero will be evaluated during the statistical testing step.

Instead of using a *linear* model, `DESeq2` and `edgeR` rely on a *negative binomial* model to fit the observed read counts to arrive at the estimate for the difference. Originally, read counts had been modeled using the *Poisson* distribution because:

- individual reads can be interpreted as binary data (Bernoulli trials): they either originate from gene i or not.
- we are trying to model the discrete probability distribution of the number of successes (success = read is present in the sequenced library).
- the pool of possible reads that could be present is large, while the proportion of reads belonging to gene i is quite small.

The convenient feature of a Poisson distribution is that *variance = mean*. Thus, if the RNA-seq experiment gives us a precise estimate of the mean read counts per condition, we implicitly know what kind of variance to expect for read counts that are not truly changing between two conditions. This, in turn, then allows us to identify those genes that show greater differences between the two conditions than expected by chance.

While read counts of the same library preparation (= technical replicates) can indeed be well approximated by the Poisson distribution, it has been shown that biological replicates have greater variance (noise) than expected. This *overdispersion* can be captured with the *negative binomial* distribution, which is a more general form of the Poisson distribution that allows the variance to exceed the mean. The square root of the dispersion is the coefficient of variation – $\frac{SD}{mean}$ – after subtracting the variance we expect due to Poisson sampling.

In contrast to the Poisson distribution, we now need to estimate two parameters from the read counts: the mean as well as the dispersion. The precision of these estimates strongly depends on the number (and variation) of replicates – the more replicates, the better the grasp on the underlying mean expression values of unchanged genes and the variance that is due to biological variation rather than the experimental treatment. For most RNA-seq experiments, only two to three replicates are available, which is not enough for reliable mean and variance estimates. Some tools therefore compensate for the lack of replication by borrowing information across genes with similar expression values and shrink a given gene's variance towards the regressed values. These fitted values of the mean and dispersion are then used instead of the raw estimates to test for differential gene expression.

6.2 Testing the null hypothesis

The null hypothesis is that there is no systematic difference between the average read count values of the different conditions for a given gene. Which test is used to assign a p-value again depends on the tool (Table 7), but generally you can think of them as some variation of the well-known t−-test (How dissimilar

*In plain English: rlog-normalized expression values for gene X modeled based on the genotype.

are the means of two populations?) or ANOVAs (How good does a reduced model capture the data when compared to the full model with all coefficients?).

Once you've obtained a list of p-values for all the genes of your data set, it is important to realize that you just performed the same type of test for thousands and thousands of genes. That means, that even if you decide to focus on genes with a p-value smaller than 0.05, if you've looked at 10,000 genes your final list may contain $0.05 * 10,000 = 500$ false positive hits. To guard yourself against this, all the tools will offer some sort of correction for the multiple hypotheses you tested, e.g. in the form of the Benjamini-Hochberg formula. You should definitely rely on the adjusted p-values rather than the original ones to zoom into possible candidates for downstream analyses and follow-up studies.

6.3 Running DGE analysis tools

6.3.1 DESeq2 workflow

For our example data set, we would like to compare the effect of the *snf2* mutants versus the wildtype samples, with the wildtype values used as the denominator for the fold change calculation.

```
# DESeq uses the levels of the condition to determine the order of the
   comparison
> str(colData(DESeq.ds)$condition)

# set WT as the first-level-factor
> colData(DESeq.ds)$condition <- relevel(colData(DESeq.ds)$condition, "WT")
```

Now, running the DGE analysis is very simple:

```
> DESeq.ds <- DESeq(DESeq.ds)
```

The `DESeq()` function is basically a wrapper around the following three individual functions:

```
> DESeq.ds <- estimateSizeFactors(DESeq.ds) # sequencing depth normalization
   between the samples
> DESeq.ds <- estimateDispersions(DESeq.ds) # gene-wise dispersion estimates
   across all samples
> DESeq.ds <- nbinomWaldTest(DESeq.ds) # this fits a negative binomial GLM and
   applies Wald statistics to each gene
```

! Note that the DGE analysis is performed on the *raw* read counts (untransformed, not normalized for sequencing depth), so supplying anything but raw read counts to either `DESeq` or `edgeR` will result in nonsensical results.

The `results()` function lets you extract the base means across samples, moderated log_2 fold changes, standard errors, test statistics etc. for every gene.

```
> DGE.results <- results(DESeq.ds, independentFiltering = TRUE, alpha = 0.05)
> summary(DGE.results)

# the DESeqResult object can basically be handled like a data.frame
> head(DGE.results)
> table(DGE.results$padj < 0.05)
> rownames(subset(DGE.results, padj < 0.05))
```

6.3.2 Exploratory plots following DGE analysis

Histograms Histograms are a simple and fast way of getting a feeling for how frequently certain values are present in a data set. A common example is a histogram of p-values (Figure 19).

```
> hist(DGE.results$pvalue,
     col = "grey", border = "white", xlab = "", ylab = "",
     main = "frequencies of p-values")
```

MA plot MA plots were originally developed for microarray visualization, but they are also useful for RNA-seq analyses. The MA plot provides a global view of the relationship between the expression change between conditions (log ratios, M), the average expression strength of the genes (average mean, A) and the ability of the algorithm to detect differential gene expression: genes that pass the significance threshold (adjusted p-value <0.05) are colored in red (Figure 19).

```
> plotMA(DGE.results, alpha = 0.05, main = "WT vs. SNF2 mutants",
         ylim = c(-4,4))
```

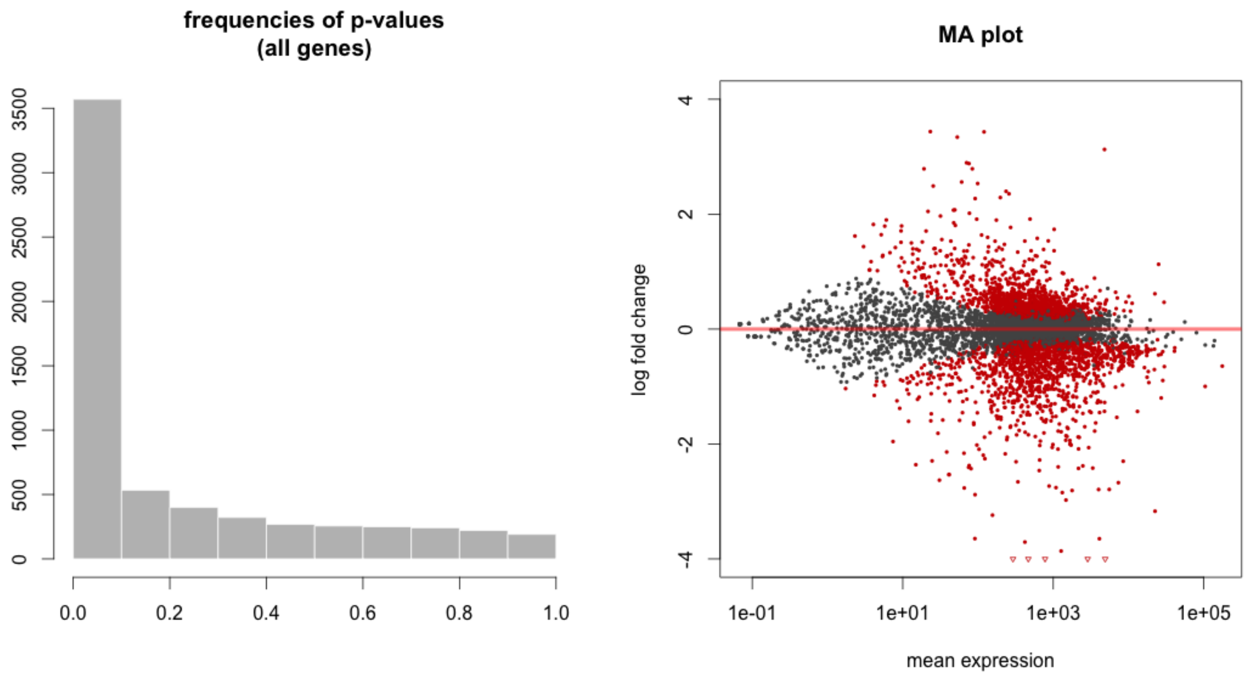


Figure 19: Left: Histogram of *p*-values for all genes tested for no differential expression between the two conditions, SNF2 and WT. Right: The MA plot shows the relationship between the expression change (M) and average expression strength (A); genes with adjusted *p*-values <0.05 are marked in red.

Heatmaps Heatmaps are a popular means to visualize the expression values across the individual samples. The following commands can be used to obtain heatmaps for *rlog*-normalized read counts for genes that show differential expression with adjusted p-values <0.05. Note that there are numerous functions for generating heatmaps in R; here we will use a fairly new function called `aheatmap()`, which is similar to `gplots::heatmap.2()` and `pheatmap::pheatmap()`.

```
# load the library with the aheatmap() function
> library(NMF)

# aheatmap needs a matrix of values, e.g., a matrix of DE genes with the
  transformed read counts for each replicate
# sort the results according to the adjusted p-value
> DGE.results.sorted <- DGE.results[order(DGE.results$padj), ]

# identify genes with the desired adjusted p-value cut-off
> DGEgenes <- rownames(subset(DGE.results.sorted, padj < 0.05))
```

© 2015-2018 Applied Bioinformatics Core | Weill Cornell Medical College

```

# extract the normalized read counts for DE genes into a matrix
> hm.mat_DGEgenes <- log.norm.counts[DGEgenes, ]

# plot the normalized read counts of DE genes sorted by the adjusted p-value
> aheatmap(hm.mat_DGEgenes, Rowv = NA, Colv = NA)

# combine the heatmap with hierarchical clustering
> aheatmap(hm.mat_DGEgenes,
    Rowv = TRUE, Colv = TRUE, # add dendrograms to rows and columns
    distfun = "euclidean", hclustfun = "average")

# scale the read counts per gene to emphasize the sample-type-specific
   differences
> aheatmap(hm.mat_DGEgenes,
          Rowv = TRUE, Colv = TRUE,
          distfun = "euclidean", hclustfun = "average",
          scale = "row") # values are transformed into distances from the center
               of the row-specific average: (actual value - mean of the group) /
              standard deviation
```

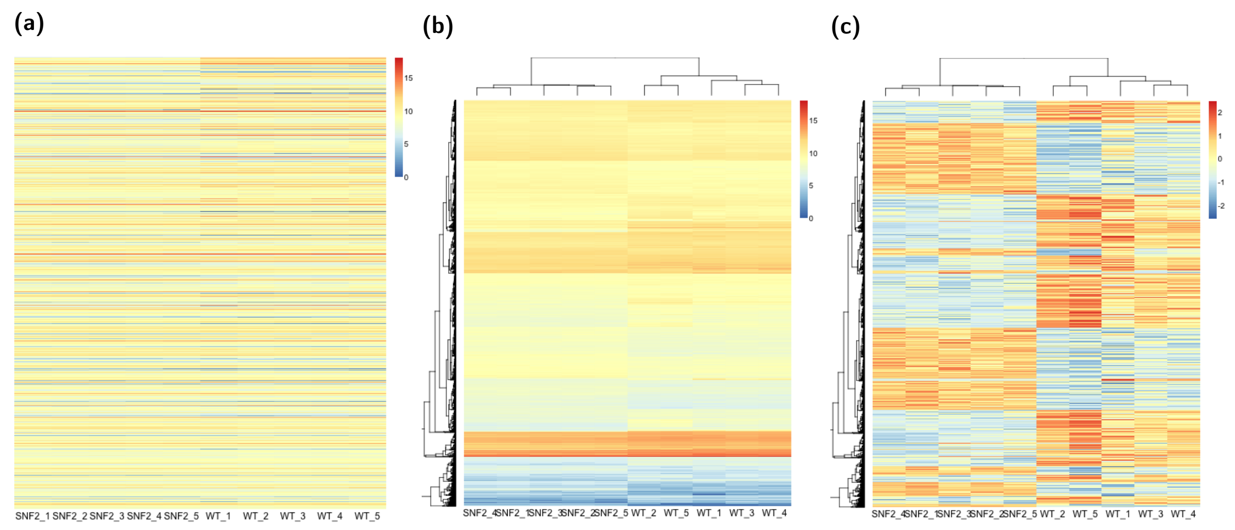


Figure 20: Heatmaps of *rlog*-transformed read counts for genes with adjusted p-values <0.05 in the DGE analysis. a) Genes sorted according to the adjusted p-values of the DGE analysis. b) Genes sorted according to hierarchical clustering. c) Same as for (b), but the read count values are scaled per row so that the colors actually represent z-scores rather than the underlying read counts.

Read counts of single genes An important sanity check of your data and the DGE analysis is to see whether genes about which you have prior knowledge behave as expected. For example, the samples named "SNF2" were generated from a mutant yeast strain where the *snf2* gene was deleted, so *snf2* should be among the most strongly downregulated genes in this DGE analysis.

To check whether *snf2* expression is absent in the mutant strain, we first need to map the ORF identifiers that we used for generating the read count matrix to the gene name so that we can retrieve the *rlog*-transformed read counts and the moderated log_2 fold changes. There is more than one way to obtain annotation data, here we will use a data base that can be directly accessed from within R. The website `https://www.bioconductor.org/packages/release/data/annotation/` lists all annotation packages that are available through `bioconductor`. For our yeast samples, we will go with `org.Sc.sgd.db`. For human data you could, for example, use `org.Hs.eg.db`.

```
> source("http://bioconductor.org/biocLite.R")
> biocLite("org.Sc.sgd.db")
```

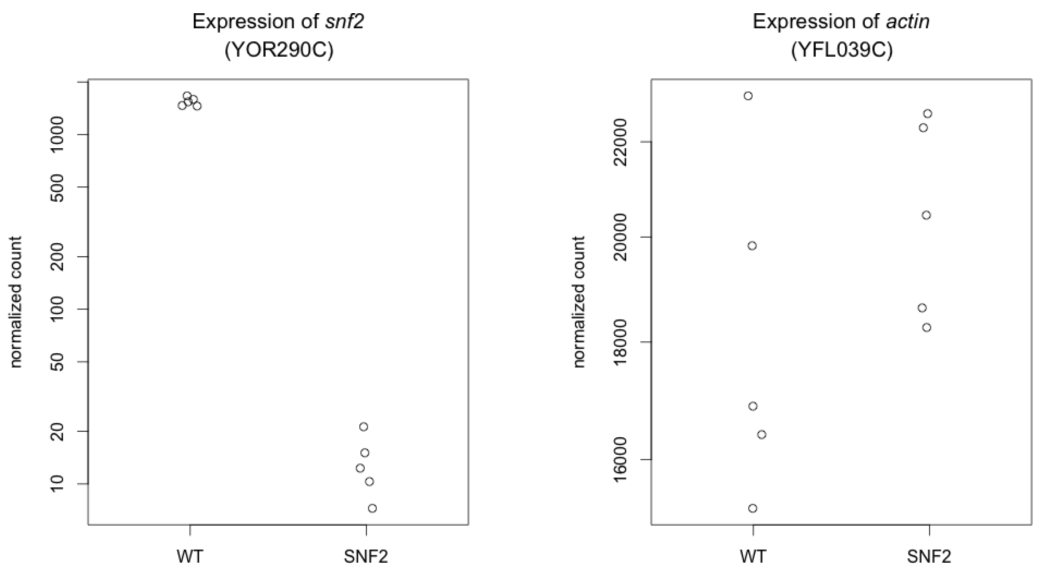


Figure 21: Read counts for *snf2* and *actin* in the replicates of both conditions.

```
> library(org.Sc.sgd.db)

# list the words (keytypes) that are available to query the annotation data
   base
> keytypes(org.Sc.sgd.db)

# list columns that can be retrieved from the annotation data base
> columns(org.Sc.sgd.db)

# make a batch retrieval for all DE genes
> anno <- select(org.Sc.sgd.db,
               keys = DGEgenes, keytype = "ORF", # to retrieve all genes: keys
                  = keys(org.Sc.sgd.db)
               columns = c("SGD","GENENAME","CHR"))

# check whether SNF2 pops up among the top downregulated genes
> DGE.results.sorted_logFC <- DGE.results[order(DGE.results$log2FoldChange), ]
> DGEgenes_logFC <- rownames(subset(DGE.results.sorted_logFC, padj < 0.05))
> head(anno[match(DGEgenes_logFC, anno$ORF), ])

# find the ORF corresponding to SNF2
subset(anno, GENENAME == "SNF2")

# DESeq2 offers a wrapper function to plot read counts for single genes
> library(grDevices) # for italicizing the gene name
> plotCounts(dds = DESeq.ds,
           gene = "YOR290C",
           normalized = TRUE, transform = FALSE,
           main = expression( atop("Expression of "*italic("snf2"), "(YOR290C)"
              )) )
```

While R offers myriad possibilities to perform downstream analyses on the lists of DE genes, you may also need to export the results into a simple text file that can be opened with other programs.

```
# merge the information of the DGE analysis with the information about the
   genes
> out.df <- merge(as.data.frame(DGE.results), anno,
        by.x = "row.names", by.y = "ORF")

# export all values for all genes into a tab-separated text file
> write.table(out.df, file = "DESeq2results_WT-vs-SNF2.txt",
      sep = "\t", quote = FALSE, row.names = FALSE)
```

?

1. What is the difference between the moderated *log*-transformed values reported by either `rlog(DESeqDataSet)` or `results(DESeqDataSet)`?
2. Which analyses should be preferably performed on *log*-transformed read counts, which ones on *log* fold changes?

6.3.3 Exercise suggestions

The following exercises will help you to familiarize yourself with the handling of the data objects generated by `DESeq2`:

1. Make a heatmap with the 50 genes that show the strongest change between the conditions. (the cut-off for the adjusted p-value should remain in place)
2. Plot the read counts for a gene that is not changing between the two conditions, e.g., *actin*.
3. Write a function that will plot the *rlog*-transformed values for a single gene, as in Figure 21. (Hint: aim for a boxplot via `plot()`, then add individual dots via `points()`)

6.3.4 edgeR

`edgeR` is very similar in spirit to `DESeq2`: both packages rely on the negative binomial distribution to model the raw read counts in a gene-wise manner while adjusting the dispersion estimates based on trends seen across all samples and genes (Table 7). The methods are, however, not identical, and results may vary. The following commands should help you perform a basic differential gene expression analysis, analogous to the one we have shown you for `DESeq2`, where five replicates from two conditions ("SNF2", "WT") were compared.

```
# install edgeR from bioconductor
> source("http://www.bioconductor.org/biocLite.R")
> biocLite("edgeR")
> library(edgeR)
```

`edgeR` requires a matrix of read counts where the row names = gene IDs and the column names = sample IDs. Thus, we can use the same object that we used for `DESeq2`.

```
> head(read.counts)
```

In addition, we need to specify the sample types, similarly to what we did for `DESeq2`.

```
> sample_info.edger <- factor(c( rep("WT", 5), rep("SNF2", 5)))
> sample_info.edger <- relevel(sample_info.edger, ref = "WT")
```

Now, `DGEList()` is the function that converts the count matrix into an `edgeR` object.

```
> edgeR.DGElist <- DGEList(counts = readcounts, group = sample_info.edger)

# check the result
> head(edgeR.DGElist$counts)
> edgeR.DGElist$samples
```

`edgeR` also recommends removing genes with almost no coverage. In order to determine a sensible cutoff, we plot a histogram of counts per million calculated by `edgeR`'s `cpm()` function.

```
# get an impression of the coverage across samples
> hist(log2(rowSums(cpm(edgeR.DGElist))))
> summary(log2(rowSums(cpm(edgeR.DGElist))))
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  2.927   8.270   9.366   9.235  10.440  17.830
```

For this data set, we're going to focus on genes that have at least one count per million reads in at least five of the ten samples.

```
# remove genes that do not have one count per million in at least 5 samples
# (adjust this to your sample!)
> keep <- rowSums( cpm(edgeR.DGElist) >= 1) >= 5
> edgeR.DGElist <- edgeR.DGElist[keep,]

# recompute library sizes after filtering
> edgeR.DGElist$samples$lib.size <- colSums(edgeR.DGElist$counts)
> head(edgeR.DGElist$samples)
```

Calculate normalization factors for the library sizes. We use the standard `edgeR` method here, which is the trimmed mean of M-values; if you wanted to use, for example, `DESeq`'s size factor, you could use `method = "RLE")`. See Table 13 for details of the methods.

```
> edgeR.DGElist <- calcNormFactors(edgeR.DGElist, method = "TMM")
> edgeR.DGElist$samples
```

To determine the differential expression in a gene-wise manner, `edgeR` first estimates the dispersion and subsequently tests whether the observed gene counts fit the respective negative binomial model. Note that the following commands are only appropriate if your data is based on an experiment with a single factor (e.g., mouse strain A vs. B; untreated cell culture vs. treated cell culture). For details on more complicated experimental set-ups, see the vignette of `edgeR` which can be found at the `bioconductor` website.

```
# specify the design setup - the design matrix looks a bit intimitating, but if
# you just focus on the formula [~sample_info.edger] you can see that it's
# exactly what we used for DESeq2, too
> design <- model.matrix(~sample_info.edger)

# estimate the dispersion for all read counts across all samples
> edgeR.DGElist <- estimateDisp(edgeR.DGElist, design)

# fit the negative binomial model
> edger_fit <- glmFit(edgeR.DGElist, design)

# perform the testing for every gene using the neg. binomial model
> edger_lrt <- glmLRT(edger_fit)
```

In contrast to `DESeq`, `edgeR` does not produce any values similar to the *rlog*-transformed read count values. You will, however, get library-size normalized log_2 fold changes.

```
# extract results from edger_lrt$table plus adjusted p-values
> DGE.results_edgeR <- topTags(edger_lrt, n = Inf, # to retrieve all genes
              sort.by = "PValue", adjust.method = "BH")
```

6.3.5 `limma-voom`

`Limma` was originally developed for the analysis of microarray gene expression data using linear models. The functions of the `limma` package have continuously been developed for much more than a decade and have laid the foundation for many widely used statistical concepts for differential gene expression analysis (Smyth, 2004). In order to use the functionalities that had specifically been developed for microarray-based data, Law et al. (2014) implemented "precision weights" that are meant to transform the finicky count data (with all its statistically annoying properties including heteroskedasticity shown in Figure 16) into more tractable normally distributed data. The two main differences to `edgeR` and `DESeq` are:

- count values are transformed to log-cpm;
- instead of negative binomial models, linear models are used (on the log-cpm values plus "precision weights").

The steps `limma` takes are:

© 2015-2018 Applied Bioinformatics Core | Weill Cornell Medical College

1. For every sample and gene, calculate the counts per million reads and log-transform these.
2. Fit a linear model to the log-cpm values taking the experimental design into account (e.g., conditions, batches etc.).
3. Use the resulting residual standard deviations for every gene to fit a global mean-variance trend across all genes and samples.
4. To obtain a "precision weight" for *single* gene observation (i.e., for every sample!), the fitted log-cpm values from step 2 are used to predict the counts for every gene and every sample. The mean-variance trend (step 3) is then used to interpolate the corresponding standard deviation for these predicted counts.
5. The squared inverse of this observation-wise estimated standard deviation is used as a penalty (inverse weight) during the test for differential expression. These penalty values are the above mentioned "precision weights".

Like `DESeq` and `edgeR`, `limma` starts with a matrix of raw read counts where each gene is represented by a row and the columns represent samples. `limma` assumes that rows with zero or very low counts have been removed. In addition, size factors for sequencing depth can be calculated using `edgeR`'s `calcNormFactors()` function.

```
> library(edgeR)
# use edgeR to remove lowly expressed genes and normalize reads for
# sequencing depth; see code chunks above
# > sample_info.edger <- factor(c( rep("SNF2", 5), rep("WT", 5)))
# > sample_info.edger <- relevel(sample_info.edger, ref = "WT")
# > edgeR.DGElist <- DGEList(counts = readcounts, group = sample_info.edger)
# > keep <- rowSums( cpm(edgeR.DGElist) >= 1) >= 5
# > edgeR.DGElist <- edgeR.DGElist[keep,]
# > edgeR.DGElist <- calcNormFactors(edgeR.DGElist, method = "TMM")
```

```
> library(limma)

# limma also needs a design matrix, just like edgeR
> design <- model.matrix(~sample_info.edger)

# transform the count data to log2-counts-per-million and estimate
# the mean-variance relationship, which is used to compute weights
# for each count -- this is supposed to make the read counts
# amenable to be used with linear models
> design <- model.matrix(~sample_info.edger)
> rownames(design) <- colnames(edgeR.DGElist)
> voomTransformed <- voom(edgeR.DGElist, design, plot=FALSE)

# fit a linear model for each gene
> voomed.fitted <- lmFit(voomTransformed, design = design)

# compute moderated t-statistics, moderated F-statistics,
# and log-odds of differential expression
> voomed.fitted <- eBayes(voomed.fitted)

# extract gene list with logFC and statistical measures
> colnames(design) # check how the coefficient is named
> DGE.results_limma <- topTable(voomed.fitted, coef = "sample_info.edgerSNF2",
                                number = Inf, adjust.method = "BH",
                                sort.by = "logFC")
```

From the limma user manual:

> The `logFC` column gives the value of the contrast. Usually this represents a log2-fold change between two or more experimental conditions although sometimes it represents a log2-expression

level. The `AveExpr` column gives the average log2-expression level for that gene across all the arrays and channels in the experiment. Column `t` is the moderated t-statistic. Column `P.Value` is the associated p-value and `adj.P.Value` is the p-value adjusted for multiple testing. The most popular form of adjustment is "BH" which is Benjamini and Hochberg's method to control the false discovery rate. (...) The B-statistic (lods or `B`) is the log-odds that the gene is differentially expressed. Suppose for example that $B = 1.5$. The odds of differential expression is $exp(1.5) = 4.48$, i.e, about four and a half to one. The probability that the gene is differentially expressed is $4.48/(1+4.48) = 0.82$, i.e., the probability is about 82% that this gene is differentially expressed. A B-statistic of zero corresponds to a 50-50 chance that the gene is differentially expressed. The B-statistic is automatically adjusted for multiple testing by assuming that 1% of the genes, or some other percentage specified by the user in the call to eBayes(), are expected to be differentially expressed. The p-values and B-statistics will normally rank genes in the same order. In fact, if the data contains no missing values or quality weights, then the order will be precisely the same.

6.4 Judging DGE results

Once you have obtained a table of genes that show signs of differential expression, you have reached one of the most important milestones of RNA-seq analysis! To evaluate how confident you can be in that list of DE genes, you should look at several aspects of the analyses you did and perform basic checks on your results:

1. Did the unsupervised clustering and PCA analyses reproduce the major trends of the initial experiment? For example, did replicates of the same condition cluster together and were they well separated from the replicates of the other condition(s)?
2. How well do different DGE programs agree on the final list of DE genes? You may want to consider performing downstream analyses only on the list of genes that were identified as DE by more than one tool.
3. Do the results of the DGE analysis agree with results from small-scale experiments? Can you reproduce qPCR results (and vice versa: can you reproduce the results of the DGE analysis with qPCR)?
4. How robust are the observed fold changes? Can they explain the effects you see on a phenotypic level?

! If your RNA-seq results are suggesting expression changes that differ dramatically from everything you would have expected based on prior knowledge, you should be very cautious!

The following code and images should just give you some examples of typical follow-up visualizations that can be done. To avoid having to load full libraries, we will use the syntax R `library::function`, i.e., `gplots::venn` uses the function `venn()` of the `gplots` package. This will directly call that function without loading all other functions of `gplots` into the working environment.

```
# make a Venn diagram
> DE_list <- list(edger = rownames(subset(DGE.results_edgeR$table, FDR<=0.05)),
                  deseq2 = rownames(subset(DGE.results, padj<=0.05)),
                  limma = rownames(subset(DGE.results_limma, adj.P.Val<=0.05)))
> gplots::venn(DE_list)

# more sophisticated venn alternative, especially if you are comparing more
    than 3 lists
> DE_gns <- UpSetR::fromList(DE_list)
> UpSetR::upset(DE_gns, order.by = "freq")

# correlation of logFC for genes found DE in all three tools
> DE_gns_all <- row.names(DE_gns[rowSums(DE_gns) == 3,]) # extract the names
# make a data.frame of fold change values
> DE_fc <- data.frame(edger = DGE.results_edgeR[DE_gns_all,]$table$logFC,
                      limma = DGE.results_limma[DE_gns_all,]$logFC,
                      deseq2 = DGE.results[DE_gns_all,]$log2FoldChange,
                         row.names = DE_gns_all)
# visually check how well the estimated logFC of the different tools agree for
   the DE genes
> pairs(DE_fc)

# heatmap of logFC
> pheatmap::pheatmap( as.matrix(DE_fc) )
```

See Figure 22 to see the plots generated by the code above. They illustrate that there's a large agreement between the three different tools since the vast majority of genes is identified by all three tools as differentially expressed. More importantly, all three tools agree on the direction and the magnitude of the fold changes although there are some individual genes where `DESeq2`'s estimates of the log fold changes are slightly different than the ones from `edgeR` or `limma`.

6.5 Example downstream analyses

While the following list is by no means exhaustive, it may give you an idea of additional possible directions and tools to explore.

- RNA-seq with **more complex experimental designs** – the vignettes of `DESeq2` and `edgeR` have a good introduction and examples, e.g., for time course experiments, paired samples etc. as well as filtering genes with the `genefilter` package (Bourgon et al., 2010). For a very comprehensive description of how the theories of linear models and particularly numerous motivations about the design matrix, have a look at Smyth (2004).
- GO term enrichment analysis
  - `goana()` function of the `limma` package
  - `goseq` R package (statistically the same as `goana()`. but with more pre-loaded annotations)
  - GOrilla (`http://cbl-gorilla.cs.technion.ac.il/`)
  - REVIGO (`http://revigo.irb.hr/`)
  - g:profiler (`http://biit.cs.ut.ee/gprofiler/`)
  - Gene Set Enrichment Analysis (GSEA; `https://www.broadinstitute.org/gsea/index.jsp`)
  - Ingenuity Pathway Analysis Studio (commercial software)
  - ...

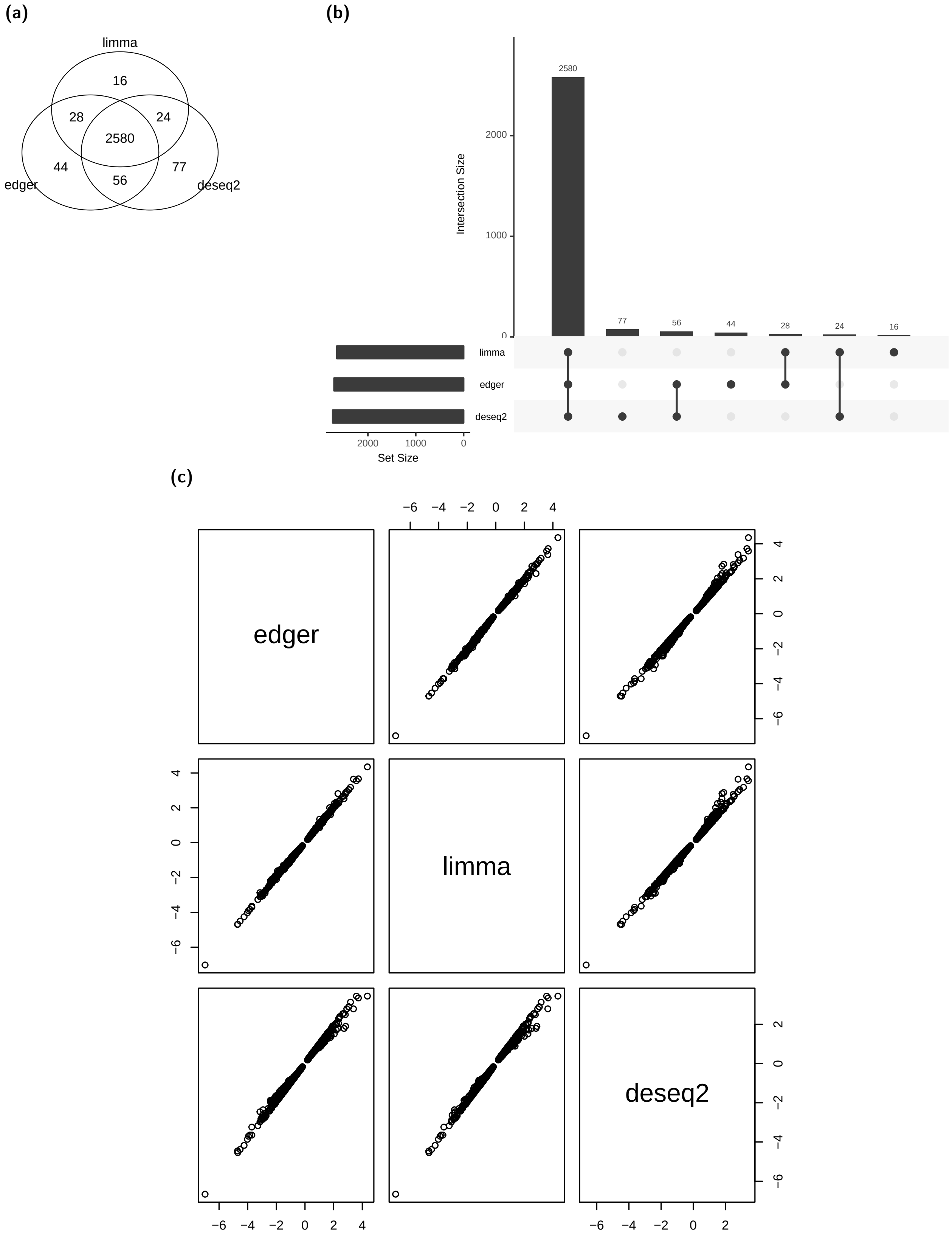


Figure 22: Some basic plots to judge the agreement of the three different DGE tools that we used. **(a)** Venn diagram of gene names. **(b)** Upset plot of gene names that displays the total size of every set in the bottom left corner, followed by the type of intersection (dots connected by lines) and the size of the intersection using vertical bars. **(c)** Pairwise plots of estimated/moderated log-fold-changes as determined by either one of the tools. Shown here are the genes that were identified as DE in all three tools (2,580 genes). The code used to generate those images is shown at the beginning of section 6.4.

7 Appendix

7.1 Improved alignment

The problem with our alignment command in Section 3.2 is that the reads contain massive insert sizes. This is most likely due to the settings controlling the size of what an acceptable intron looks like. Since yeast has a fairly small genome with relatively few (and small) introns per gene, we should tune that parameter to fit the specific needs.

To determine suitable lower and upper limits for intron sizes, we will need to download an annotation that will allow us to determine those sizes easily. This could, for example, be done via the UCSC Table Browser, as described in this Biostars post: `https://www.biostars.org/p/13290/`.

```
# get min. intron size
$ awk '{print $3-$2}' introns_yeast.bed | sort -k1n | uniq | head -n 3
1
31
35

# get max. intron size
$ awk '{print $3-$2}' introns_yeast.bed | sort -k1n | uniq | tail -n 3
1623
2448
2483
```

Now that we have a feeling for what the sizes of annotated introns look like, we can re-run `STAR`:

```
runSTAR=~/mat/software/STAR-2.5.3a/bin/Linux_x86_64/STAR
REF_DIR=~/mat/referenceGenomes/S_cerevisiae/STARindex/

for SAMPLE in WT_1 SNF2_1
do
  # get a comma-separated list of fastq files for each sample
  for FASTQ in ~/mat/precomputed/rawReads_yeast_Gierlinski/${SAMPLE}/*fastq.gz
  do
    FILES=`echo $FASTQ,$FILES` # this will have an additional comma in the end
  done

  FILES=`echo $FILES | sed 's/,$//'` # if you want to remove the last comma

  echo "Aligning files for ${SAMPLE}, files:"
  echo $FILES

  $runSTAR --genomeDir ${REF_DIR} --readFilesIn $FILES \
  --readFilesCommand gunzip -c  \
  --outFileNamePrefix ${SAMPLE}_  \
  --outFilterMultimapNmax 1 \
  --outSAMtype BAM SortedByCoordinate \
  --runThreadN 2 \
  --twopassMode Basic  \
  --alignIntronMin 1  \
  --alignIntronMax 2500
done
```

7.2 Additional tables

Table 8: All high-throughput sequencing data will suffer from some degree of bias due to the biochemistry of the sequencing, the detection technique and bioinformatics processing. Biases that are oftentimes sample-specific (e.g., GC content, fragment length distributions) are common sources of technical variation that can either mask or (worse!) mimick biological signals. For descriptions of RNA-seq specific biases, see the main text and ('t Hoen et al., 2013; Li et al., 2014; Su et al., 2014).

Problem	Reasons	Solutions
Batch effects	• variation in the sample processing (e.g., reagent batches, experimenters, pipetting accuracy) • flowcell inconsistencies • differences between sequencing runs (e.g., machine calibration)	• appropriate experimental design (e.g., proper randomization (Auer and Doerge, 2010; Honaas et al., 2016)) • samples of the same experiment should have similar quality and quantity • optimal experimental conditions (use of master mixes etc.)
Library preparation (PCR- -dependent biases)	• varying *GC content* can result in very distinct, library-specific fragment yields • *fragment size*: small fragments are preferably hybridized to the flowcell • low number of founder DNA fragments will yield numerous *duplicated fragments*	• optimizing cross-linking, sonication, and the mRNA enrichment to ensure that the majority of the transcriptome is present in the sample • limiting PCR cycles during library preparation to a minimum • computational correction for GC content and elimination of reads from identical DNA fragments (e.g., (Benjamini and Speed, 2012))
Sequencing errors and errors in base calling	• loss of synchronized base incorporation into the single molecules within one cluster of clonally amplified DNA fragments (phasing and pre-phasing) • mixed clusters • signal intensity decay over time due to unstable reagents • uneven signal intensities depending on the position on the flowcell • overlapping emission frequency spectra of the four fluorescently-labelled nucleotides	• improvement of the sequencing chemistry and detection • optimized software for base calling • computational removal of bases with low base calling scores
Copy number variations and mappability	• incomplete genome assemblies • strain-specific differences from the reference assembly may lead to misrepresentation of individual loci • repetitiveness of genomes and shortness of sequencing reads hinder unique read alignment	• longer sequencing reads • paired-end sequencing • exclusion of blacklisted regions that are known to attract artificially high read numbers (Kundaje, 2013) • computational correction for mappability

Table 9: FASTQC modules.

Plot title	**Details**	**Warning (Failure)**	**Solution**
Per *base* sequence quality	This plot is based on the quality scores reported and stored by the sequencing platform (Phred scores, see above). For Illumina sequencing, the reagents degrade throughout the sequencing run which is why the last bases tend to get worse scores.	Lower quartile for any base <10 (<5), or median for any base <25 (<20).	Trimming reads based on their average quality.
Per *tile* sequence quality	Check whether a part of the flowcell failed (e.g., due to bubbles or dirt on the flowcell). The plot shows the deviation from the average quality.	Any tile with a mean Phred score >2 (>5) for that base across all tiles.	The `FASTQ` file contains the tile IDs for each read which can be used to filter out reads from flawed tiles. Note though that variation in the Phred score of a flowcell is also a sign of overloading; tiles that are affected for only few cycles can be ignored.
Per *sequence* quality scores	Identify putative subsets of poor sequences.	Most frequent mean quality <27 (<20).	Quality trimming.
Per base sequence content	Expectation: all bases should be sequenced as often as they are represented in the genome. Reasons why this assumption may not hold: • random hexamer priming during library preparation • contamination (e.g., adapter dimers) • bisulfite treatment (loss of cytosines)	Difference between A and T, or G and C >10% (>20%) in any position.	If overrepresented sequences are the cause of a failure, these can be removed.
Per sequence GC content	The GC content of each read should be roughly normally distributed (maximum should correspond to the average GC content of the genome). Note that the reference shown here is also based on the supplied `FASTQ` file, therefore it will not be able to detect a global shift of GC content.	Deviations from the normal distribution for >15% (30%) of the reads	Sharp peaks on an otherwise smooth distribution usually indicate a specific contaminant that should also be reported as an overrepresented sequence. Broader peaks outside the expectated distribution may represent contaminating sequences from a species with a different GC genome content.
Per base N content	Percentage of base calls at each position for which an `N` was called by the sequencer indicating lack of confidence to make a base call.	Any position with an `N` content of >5% (20%)	A common reason for large numbers of `N` is lack of library complexity, i.e., if all reads start with the same bases, the sequencer may not be able to differentiate them sufficiently.

Continued on next page

Table 9 – *Continued from previous page*

Plot title	**Details**	**Warning (Failure)**	**Solution**
Sequence length distribution	Determines the lengths of the sequences of the `FASTQ` file. The distribution of read sizes should be uniform for Illumina sequencing.	Warning is raised if not all sequences are the same length; failure is issued if any sequence has zero length.	For Illumina sequencing, different read lengths should only occur if some sort of bioinformatic trimming has happened prior to the FASTQC analysis.
Duplicate sequences	The sequenced library should contain a random and complete representation of the genome or transcriptome, i.e., most sequences should occur only once. High duplication rates are indicative of PCR overamplification which may be caused by an initial lack of starting material. Note that this module only takes the first 100,000 sequences of each file into consideration.	Non-unique sequences make up more than 20% (50%) of all reads.	Unless you have reasons to expect sequences to be duplicated (i.e., specific enrichments of certain sequences), this plot is a strong indicator of suboptimal sample preparation. Duplication due to excessive sequencing will be reflected by flattened lines in the plot; PCR overamplification of low complexity libraries are indicated by sharp peaks towards the right-hand side of the plot.
Over-represented sequences	All sequences which make up more than 0.1% of the first 100,000 sequences are listed.	Any sequence representing >0.1% (1%) of the total.	Bioinformatic removal of contaminating sequences.
Adapter content	This module specifically searches for a set of known adapters. The plot itself shows a cumulative percentage count of the proportion of your library which has seen each of the adapter sequences at each position.	Any adapter present in more than 5% (10%) of all reads.	Bioinformatic removal of adapter sequences.
K-mer content	The number of each 7-mer at each position is counted; then a binomial test is used to identify significant deviations from an even coverage at all positions (only 2% of the whole library is analyzed and the results are extrapolated).	Any k-mer overrepresented with a binomial p-value <0.01 ($<10^{-5}$)	Libraries which derive from random priming will nearly always show k-mer bias at the start of the library due to an incomplete sampling of the possible random primers. Always check the results of the adapter content and overrepresented sequences modules, too.

Table 10: Types of optional entries that can be stored in the header section of a `SAM` file following the format `@<Section> <Tag>:<Value>`. The asterisk indicates which tags are required if a section is set.

Section	Tag	Description
HD (header)	VN*	File format version
	SO	Sort order ("unsorted", "queryname", or "coordinate")
SQ (sequence dictionary)	SN*	Sequence name (e.g., chromosome name)
	LN*	Sequence length
	AS	Genome assembly identifier (e.g., mm9)
	M5	MD5 checksum of the sequence
	UR	URI of the sequence
	SP	Species
RG (read group)	ID*	Read group identifier (e.g. "Lane1")
	SM*	Sample
	LB	Library
	DS	Description
	PU	Platform unit (e.g., lane identifier)
	PI	Predicted median insert size
	CN	Name of the sequencing center
	DT	Date of the sequencing run
	PL	Sequencing platform
PG (program)	ID*	Name of the program that produced the `SAM` file
	VN	Program version
	CL	Command line used to generate the `SAM` file
CO (comment)		Unstructured one-line comment lines (can be used multiple times)

Table 11: Overview of `RSeQC` scripts. See the online documentation of `RSeQC` (`http://rseqc.sourceforge.net`) and Wang et al. (2012) for more details. Note that tools marked in red have been shown to return erroneous results (Hartley and Mullikin, 2015).

Script	Purpose
	Basic read quality
`read_duplication`	Determine read duplication rate, based on the sequence only (`output.dup.seq.DupRate.xls`) as well as on the alignment positions (`output.dup.pos.DupRate.xls`).
`read_hexamer`	Calculates hexamer frequencies from `FASTA` or `FASTQ` files. Similar to `FASTQC`'s k-mer content analysis.
`read_GC`	Calculates % GC for all reads. Similar to `FASTQC`'s GC distribution plot, the peak of the resulting histogram should coincide with the average GC content of the genome.
`read_NVC`	Calculates the nucleotide composition across the reads; similar to `FASTQC`'s per base sequence content analysis.
`read_quality`	Calculates distributions for base qualities across reads; similar to `FASTQC`'s per base sequence quality analysis.
	Alignment QC
`bam_stat`	Calculates reads mapping statistics for a `BAM` (or `SAM`) file. Note that uniquely mapped reads are determined on the basis of the mapping quality.
`clipping_profile`	Estimates clipping profile of RNA-seq reads from BAM or SAM file. This will fail if the aligner that was used does not support clipped mapping (CIGAR strings must have `S` operation).
`mismatch_profile`	Calculates distribution of mismatches along reads based on the `MD` tag.
`insertion_profile`, `deletion_profile`	Calculates the distribution of insertions or deletions along reads.
`read_distribution`	Calculates fractions of reads mapping to transcript features such as exons, 5'UTR, 3' UTR, introns.
`RPKM_saturation`	The full read set is sequentially downsampled and RPKMs are calculated for each subset. The resulting plot helps determine how well the RPKM values can be estimated. The visualized percent relative error is calculated as $\frac{\lvert RPKM_{subsample} - RPKM_{max} \rvert}{RPKM_{max}} * 100$
	RNA-seq-specific QC
`geneBody_coverage`	Scales all transcripts to 100 bp, then calculates the coverage of each base. The read coverage should be uniform and ideally not show 5' or 3'ĂŹ bias since that would suggest problems with degraded RNA input or with cDNA synthesis.
`infer_experiment`	Speculates how RNA-seq sequencing was configured, i.e., PE or SR and strand-specificity. This is done by subsampling reads and comparing their genome coordinates and strands with those of the transcripts from the reference gene model. For non-strand-specific libraries, the strandedness of the reads and the transcripts should be independent. See `http://rseqc.sourceforge.net/#infer-experiment-py` for details.
`junction_annotation`	Compares detected splice junctions to the reference gene model, classifying them into 3 groups: annotated, complete novel, partial novel (only one of the splice sites is unannotated). The script differentiates between splice events (single read level) and splice junctions (multiple reads show the same splicing event).
`junction_saturation`	Similar concept to `RPKM_saturation`: splice junctions are detected for each sampled subset of reads. The detection of annotated splice sites should be saturated with the maximum coverage (= all supplied reads), otherwise alternative splicing analyses are not recommended because low abundance splice sites will not be detected.

Continued on next page

© 2015-2018 Applied Bioinformatics Core | Weill Cornell Medical College

Table 11 – *Continued from previous page*

Script	Purpose
`tin`	Calculates the transcript integrity number (TIN) (not to be confused with the RNA integrity number, RIN, that is calculated before sequencing based on the $28S/18S$ ratio). TIN is calculated for each transcript and represents the fraction of the transcript with uniform read coverage.
	General read file handling and processing
`bam2fq`	Converts `BAM` to `FASTQ`.
`bam2wig`	Converts read positions stored in a `BAM` file into read coverage measures all types of RNA-seq data in `BAM` format into wiggle file.
`divide_bam`	Equally divides a given `BAM` file (m alignments) into n parts. Each part contains roughly m/n alignments that are randomly sampled from the entire alignment file.
`inner_distance`	Estimates the inner distance (or insert size) between two paired RNA reads (requires PE reads). The results should be consistent with the gel size selection during library preparation.
`overlay_bigwig`	Allows the manipulation of two `BIGWIG` files, e.g., to calculate the sum of coverages. See the `-action` option for all possible operations.
`normalize_bigwig`	Normalizes all samples to the same wigsum (= number of bases covered by $read\,length * total\,no.\,of\,reads$).
`split_bam`, `split_paired_bam`	Provided with a gene list and a `BAM` file, this module will split the original `BAM` file into 3 smaller ones: 1. `*.in.bam`: reads overlapping with regions specified in the gene list 2. `*.ex.bam`: reads that do not overlap with the supplied regions 3. `*.junk.bam`: reads that failed the QC or are unaligned
`RPKM_count`	Calculates the raw count and RPKM values for each exon, intron and mRNA region defined by the provided annotation file.

Table 12: Overview of `QoRTs` QC functions. See the online documentation of `QoRTs` (`http://hartleys.github.io/QoRTs/index.html`) and Hartley and Mullikin (2015) for more details.

QC Function	Purpose
	Basic read quality
`GCDistribution`	Calculates GC content distribution for all reads. Similar to `FASTQC`'s GC distribution plot, the peak of the resulting histogram should coincide with the average GC content of the genome.
`NVC`	Calculates the nucleotide composition across the length of the reads; similar to `FASTQC`'s per base sequence content analysis.
`QualityScoreDistribution`	Calculates distributions for base qualities across reads; similar to `FASTQC`'s per base sequence quality analysis.
	Alignment QC
`chromCounts`	Calculates number of reads mapping to each category of chromosome (autosomes, allosomes, mtDNA).
`CigarOpDistribution`	Calculates rate of various CIGAR operations as a function of read length, including insertions, deletions, splicing, hard and soft clipping, padding, and alignment to reference. See Section 3.3.2 for details about CIGAR operations.
`GeneCalcs`	Calculates fractions of reads mapping to genomic features such as unique genes, UTRs, ambiguous genes, introns, and intergenic regions.
	RNA-seq-specific QC
`writeGeneBody`	Breaks up all genes into 40 equal-length counting bins and determines the number of reads that overlap with each counting bin. The read coverage should be uniform and ideally not show 5' or 3'ĂŹ bias since that would suggest degraded RNA input or problems with cDNA synthesis.
`writeGenewiseGeneBody`	Writes file containing gene-body distributions for each gene.
`StrandCheck`	Checks if the data is strand-specific by calcualting the rate at which reads appear to follow the two possible library-type strandedness rules (fr-firststrand and fr-secondstrand, described by the CuffLinks documentation at `http://cufflinks.cbcb.umd.edu/manual.html#library`).
`JunctionCalcs`	Calculates the number of novel and known splice junctions. Splice junctions are split into 4 groups, first by whether the splice junction appears in the gene annotation `GTF` (known vs. novel), and then by whether the splice junction has fewer or ≥ 4 reads covering it.
	General read file handling and processing
`InsertSize`	Estimates the inner distance (or insert size) between two paired RNA reads (requires PE reads). The results should be consistent with the gel size selection during library preparation.
`makeWiggles`	Converts read positions stored in a `BAM` file into wiggle files with 100-bp window size.
`makeJunctionBed`	Creates a `BED` file for splice-junction counts.
`writeGeneCounts`	Calculates raw number of reads mapping to each gene in the annotation file. Also creates a cumulative gene diversity plot, which shows the percent of the total proportion of reads as a function of the number of genes sequenced. This is useful as an indicator of whether a large proportion of the reads stem from of a small number of genes (as a result of ribosomal RNA or hemoglobin contamination, for example).
`calcDetailedGeneCounts`	Calculates more detailed read counts for each gene, including the number of reads mapping to coding regions, UTR, and intronic regions.

Continued on next page

Table 12 – *Continued from previous page*

Function	**Purpose**
`writeBiotypeCounts`	Write a table listing read counts for each biotype, which is a classification of genes into broader categories (e.g., protein coding, pseudogene, processed pseudogene, miRNA, rRNA, scRNA, snoRNA, snRNA). Note that, in order for this function to succeed, the optional “gene_biotype" attribute is required to be present in the gene annotation file (`GTF`).
`FPKM`	Calculates FPKM values for each gene in the annotation file.

Table 13: Normalization methods for the comparison of gene read counts between different conditions. See, for example, Bullard et al. (2010) and Dillies et al. (2013) for comprehensive assessments of the individual methods.

Name	**Details**	**Comment**
Total Count	All read counts are divided by the total number of reads (library size) and multiplied by the mean total count across all samples.	• biased by highly expressed genes • cannot account for different RNA repertoire between samples • poor detection sensitivity when benchmarked against qRT-PCR (Bullard et al., 2010)
Counts Per Million	Each gene count is divided by the corresponding library size (in millions).	• see Total Count
DESeq's size factor	1. For each gene, the **geometric mean** of read counts across all samples is calculated. 2. Every gene count is **divided by the geometric mean**. 3. A sample's size factor is the **median of these ratios** (skipping the genes with a geometric mean of zero).	• the size factor is applied to all read counts of a sample • more robust than total count normalization • implemented by the `DESeq` R library (`estimateSizeFactors()` function), also available in `edgeR` (`calcNormFactors()` function with option `method = "RLE"`) • details in Anders and Huber (2010)
Trimmed Mean of M-values (TMM)	TMM is always calculated as the weighted mean of log ratios between two samples: 1. Calculate gene-wise log_2 fold changes (= **M-values**): $M_g = log_2(\frac{Y_{gk}}{N_k})/log_2(\frac{Y_{gk'}}{N_{k'}})$ where Y is the observed number of reads per gene g in library k and N is the total number of reads. 2. **Trimming**: removal of upper and lower 30%. 3. **Precision weighing**: the inverse of the estimated variance is used to account for lower variance of genes with larger counts.	• the size factor is applied to every sample's library size; normalized read counts are obtained by dividing raw read counts by the TMM-adjusted library sizes • more robust than total count normalization • implemented in `edgeR` via `calcNormFactors()` with the default `method = "TMM"` • details in Robinson and Oshlack (2010)
Upper quartile	1. Find the upper quartile value (top 75% read counts after removal of genes with 0 reads). 2. Divide all read counts by this value.	• similar to total count normalization, thus it also suffers from a great influence of highly-expressed DE genes • can be calculated with `edgeR`'s `calcNormFactors()` function (`method = "upperquartile"`)

Table 14: Normalization methods for the comparison of gene read counts within the same sample.

Name	Details	Comment
RPKM (reads per kilobase of exons per million mapped reads)	1. For each gene, count the number of reads mapping to it (X_i). 2. Divide that count by: the length of the gene, l_i, in base pairs divided by 1,000 multiplied by the total number of mapped reads, N, divided by 10^6. $$RPKM_i = \frac{X_i}{(\frac{l_i}{10^3})(\frac{N}{10^6})}$$	• introduces a bias in the per-gene variances, in particular for lowly expressed genes (Oshlack and Wakefield, 2009) • implemented in `edgeR`'s `rpkm()` function
FPKM (fragments per kilobase...)	1. Same as RPKM, but for paired-end reads: 2. The number of fragments (defined by two reads each) is used.	• implemented in `DESeq2`'s `fpkm()` function
TPM	Instead of normalizing to the total library size, TPM represents the abundance of an individual gene i in relation to the abundances of the other transcripts (e.g., j) in the sample. 1. For each gene, count the number of reads mapping to it and divide by its length in base pairs (= counts per base). 2. Multiply that value by 1 divided by the sum of all counts per base of every gene. 3. Multiply that number by 10^6. $$TPM_i = \frac{X_i}{l_i} * \frac{1}{\sum_j \frac{X_j}{l_k}} * 10^6$$	• details in Wagner et al. (2012)

7.3 Installing bioinformatics tools on a UNIX server

Tools are shown in order of usage throughout the script. **The exact version numbers and paths may be subject to change!**

FASTQC

1. Download.

```
$ wget http://www.bioinformatics.babraham.ac.uk/projects/fastqc/fastqc_v0
    .11.7.zip
```

2. Unzip and make executable.

```
$ unzip fastqc_v0.11.7.zip
$ cd FastQC
$ chmod 755 fastqc
```

MultiQC

1. Install anaconda, a package which helps manage Python installations:

```
$ wget https://repo.continuum.io/archive/Anaconda2-5.1.0-Linux-x86_64.sh
$ bash Anaconda2-5.1.0-Linux-x86_64.sh
```

   You will need to accept the license terms, specify where to install anaconda, and specify whether you want anaconda's install location to be prepended to your `PATH` variable.

2. Make sure anaconda's install location is prepended to your `PATH` variable:

```
$ export PATH=/home/classadmin/software/anaconda2/bin:$PATH
```

3. Install MultiQC using anaconda's pip

```
$ pip install multiqc
```

samtools

1. Download source code and unzip it.

```
$ wget -O samtools-1.7.tar.bz2 https://github.com/samtools/samtools/
    releases/download/1.7/samtools-1.7.tar.bz2
$ tar jxf samtools-1.7.tar.bz2
$ cd samtools-1.7
```

2. Compile.

```
$ make
```

3. Check whether the tool is running.

```
$ ./samtools
```

4. Add the location where samtools was installed to your `PATH` variable; this way you will not need to specify the exact location everytime you want to run the tool.

```
$ export PATH=/home/classadmin/software/samtools-1.7:$PATH
```

RSeQC

1. RSeQC is a python tool like MultiQC and can be installed using pip. Make sure anaconda's install location is prepended to your `PATH` variable:

```
$ echo $PATH
# if you don't see anaconda2 somewhere in there (or not the correct path),
  do:
export PATH=/home/classadmin/software/anaconda2/bin:$PATH
```

2. Install RSeQC using anaconda's installer

```
$ pip install RSeQC
```

QoRTs

1. Install the R component (in R):

```
> install.packages("http://hartleys.github.io/QoRTs/QoRTs_LATEST.tar.gz",
                   repos=NULL,
                   type="source");
```

2. Download the Java component (in the Terminal).

```
$ wget -O qorts.jar "https://github.com/hartleys/QoRTs/releases/download
  /1.3.0/QoRTs_LATEST.tar.gz"
```

STAR

1. Download.

```
$ wget -O STAR-2.5.4b.tar.gz https://github.com/alexdobin/STAR/archive
  /2.5.4b.tar.gz
```

2. Unzip.

```
$ tar -zxf STAR-2.5.4b.tar.gz
```

To run `STAR`:

```
$ ./bin/Linux_x86_64_static/STAR
```

UCSC tools aka Kent tools

1. Figure out which operating system version you have

```
$ uname -a
```

2. Download the already compiled binaries from the corresponding folder (shown here for the Linux server) and make them executable. The programs indicated here are the ones most commonly used for typical NGS analyses, but feel free to download more (or fewer) tools.

```
$ mkdir UCSCtools
$ cd UCSCtools
$ for PROGRAM in bedGraphToBigWig bedClip bigWigAverageOverBed
  bigWigCorrelate bigWigInfo bigWigSummary faToTwoBit fetchChromSizes
  genePredToBed gff3ToGenePred liftOver wigToBigWig
 do
```

```
    wget http://hgdownload.soe.ucsc.edu/admin/exe/linux.x86_64/${PROGRAM}
    chmod +x ${PROGRAM}
  done
```

featureCounts (subread package)

1. Download.

```
$ wget --no-check-certificate https://sourceforge.net/projects/subread/
    files/subread-1.6.0/subread-1.6.0-Linux-x86_64.tar.gz
```

2. Unzip.

```
$ tar -zxf subread-1.6.0-Linux-x86_64.tar.gz
```

R

1. Download.

```
$ wget --no-check-certificate https://cran.r-project.org/src/base/R-3/R
    -3.4.3.tar.gz
```

2. Unzip.

```
$ tar zxvf R-3.4.3.tar.gz
```

3. Compile. You can use the `-prefix` option to specify the folder where you would like to install the program.

```
$ cd R-3.4.3
$ ./configure --prefix=<path to folder of choice>
$ make
$ make install
```

© 2015-2018 Applied Bioinformatics Core | Weill Cornell Medical College

References

Altman N and Krzywinski M. Points of significance: Sources of variation. *Nature Methods*, **12**(1):5–6, 2014. doi:10.1038/nmeth.3224.

Anders S and Huber W. DESeq: Differential expression analysis for sequence count data. *Genome Biology*, **11**:R106, 2010. doi:10.1186/gb-2010-11-10-r106.

Anders S, Pyl PT, and Huber W. HTSeq - A Python framework to work with high-throughput sequencing data. *Bioinformatics*, **31**(2):166–169, 2014. doi:10.1093/bioinformatics/btu638.

Anders S, Reyes A, and Huber W. Detecting differential usage of exons from RNA-seq data. *Genome Research*, **22**(10):2008–2017, 2012. doi:10.1101/gr.133744.111.

Auer PL and Doerge RW. Statistical design and analysis of RNA sequencing data. *Genetics*, **185**(2):405–16, 2010. doi:10.1534/genetics.110.114983.

Ballouz S, Dobin A, Gingeras TR, and Gillis J. The fractured landscape of RNA-seq alignment: the default in our STARs. *Nucleic Acids Research*, 2018. doi:10.1093/nar/gky325.

Benjamini Y and Speed TP. Summarizing and correcting the GC content bias in high-throughput sequencing. *Nucleic Acids Research*, **40**(10):e72, 2012. doi:10.1093/nar/gks001.

Blainey P, Krzywinski M, and Altman N. Points of Significance: Replication. *Nature Methods*, **11**(9):879–880, 2014. doi:10.1038/nmeth.3091.

Bourgon R, Gentleman R, and Huber W. Independent filtering increases detection power for high-throughput experiments. *PNAS*, **107**(21):9546–9551, 2010. doi:10.1073/pnas.0914005107.

Bray NL, Pimentel H, Melsted P, and Pachter L. Near-optimal probabilistic RNA-seq quantification. *Nat Biotech*, **34**(5):525–527, 2016. doi:10.1038/nbt.3519.

Bullard JH, Purdom E, Hansen KD, and Dudoit S. Evaluation of statistical methods for normalization and differential expression in mRNA-Seq experiments. *BMC Bioinformatics*, **11**:94, 2010. doi:10.1186/1471-2105-11-94.

Byron SA, Van Keuren-Jensen KR, Engelthaler DM, Carpten JD, and Craig DW. Translating RNA sequencing into clinical diagnostics: opportunities and challenges. *Nature Reviews Genetics*, **17**(5):257–271, 2016. doi:10.1038/nrg.2016.10.

Cathala G, Savouret JF, Mendez B, West BL, Karin M, Martial JA, and Baxter JD. A method for isolation of intact, translationally active ribonucleic acid. *DNA*, **2**(4):329–335, 1983. doi:10.1089/dna.1983.2.329.

Ching T, Huang S, and Garmire LX. Power analysis and sample size estimation for RNA-Seq differential expression. *RNA*, pp. rna.046011.114–, 2014. doi:10.1261/rna.046011.114.

Costa-Silva J, Domingues D, and Lopes FM. RNA-Seq differential expression analysis: An extended review and a software tool. *PLoS ONE*, **12**(12), 2017. doi:10.1371/journal.pone.0190152.

Dapas M, Kandpal M, Bi Y, and Davuluri RV. Comparative evaluation of isoform-level gene expression estimation algorithms for RNA-seq and exon-array platforms. *Briefings in Bioinformatics*, (Oct 2015), 2016. doi:10.1093/bib/bbw016.

Dillies MA, Rau A, Aubert J, Hennequet-Antier C, Jeanmougin M, Servant N, Keime C, Marot NS, Castel D, Estelle J, Guernec G, Jagla B, Jouneau L, Laloë D, Le Gall C, Schaëffer B, Le Crom S, Guedj M, and Jaffrézic F. A comprehensive evaluation of normalization methods for Illumina high-throughput RNA sequencing data analysis. *Briefings in Bioinformatics*, **14**(6):671–683, 2013. doi:10.1093/bib/bbs046.

Ding L, Rath E, and Bai Y. Comparison of Alternative Splicing Junction Detection Tools Using RNASeq Data. *Current Genomics*, 2017. doi:10.2174/1389202918666170215125048.

Dobin A, Davis CA, Schlesinger F, Drenkow J, Zaleski C, Jha S, Batut P, Chaisson M, and Gingeras TR. STAR: Ultrafast universal RNA-seq aligner. *Bioinformatics*, **29**(1):15–21, 2013. doi:10.1093/bioinformatics/bts635.

Dobin A and Gingeras TR. Optimizing RNA-seq mapping with STAR. In *Methods in Molecular Biology*, vol. 1415, pp. 245–262. Humana Press, New York, NY, 2016. doi:10.1007/978-1-4939-3572-7_13.

ENCODE. Standards, Guidelines and Best Practices for RNA-Seq, 2011. URL `https://genome.ucsc.edu/ENCODE/protocols/dataStandards/ENCODE_RNAseq_Standards_V1.0.pdf`.

Engström PG, Steijger T, Sipos B, Grant GR, Kahles A, Rätsch G, Goldman N, Hubbard TJ, Harrow J, Guigó R, and Bertone P. Systematic evaluation of spliced alignment programs for RNA-seq data. *Nature*

Methods, **10**(12):1185–1191, 2013. doi:10.1038/nmeth.2722.

Everaert C, Luypaert M, Maag JL, Cheng QX, DInger ME, Hellemans J, and Mestdagh P. Benchmarking of RNA-sequencing analysis workflows using whole-transcriptome RT-qPCR expression data. *Scientific Reports*, **7**(1), 2017. doi:10.1038/s41598-017-01617-3.

Ewels P, Magnusson M, Lundin S, and KÃď'ller M. Multiqc: summarize analysis results for multiple tools and samples in a single report. *Bioinformatics*, **32**(19):3047, 2016. doi:10.1093/bioinformatics/btw354.

Farrell R. *RNA methodologies laboratory guide for isolation and characterization*. Elsevier/Academic Press, Amsterdam Boston, 2010.

Feng H, Zhang X, and Zhang C. mRIN for direct assessment of genome-wide and gene-specific mRNA integrity from large-scale RNA-sequencing data. *Nature Communications*, **6**(May):7816, 2015. doi:10.1038/ncomms8816.

Genohub. Depth of Coverage (RNA), 2015. URL `https://genohub.com/next-generation-sequencing-guide/#depth2`.

Germain PL, Vitriolo A, Adamo A, Laise P, Das V, and Testa G. RNAontheBENCH: computational and empirical resources for benchmarking RNAseq quantification and differential expression methods. *Nucleic Acids Research*, **44**(11), 2016. doi:10.1093/nar/gkw448.

Gibbons FD and Roth FP. Judging the quality of gene expression-based clustering methods using gene annotation. *Genome Research*, **12**(10):1574–1581, 2002. doi:10.1101/gr.397002.

Gierliński M, Cole C, Schofield P, Schurch NJ, Sherstnev A, Singh V, Wrobel N, Gharbi K, Simpson G, Owen-Hughes T, Blaxter M, and Barton GJ. Statistical models for RNA-seq data derived from a two-condition 48-replicate experiment. *Bioinformatics*, **31**(22):1–15, 2015. doi:10.1093/bioinformatics/btv425.

Goodwin S, Mcpherson JD, and Mccombie WR. Coming of age : ten years of next- generation sequencing technologies. *Nature Genetics*, **17**(6):333–351, 2016. doi:10.1038/nrg.2016.49.

Griffith M, Walker JR, Spies NC, Ainscough BJ, and Griffith OL. Informatics for RNA Sequencing: A Web Resource for Analysis on the Cloud. *PLoS Computational Biology*, **11**(8), 2015. doi:10.1371/journal.pcbi.1004393.

Hartley SW and Mullikin JC. QoRTs: a comprehensive toolset for quality control and data processing of RNA-Seq experiments. *BMC Bioinformatics*, **16**(1):224, 2015. doi:10.1186/s12859-015-0670-5.

Head SR, Kiyomi Komori H, LaMere SA, Whisenant T, Van Nieuwerburgh F, Salomon DR, and Ordoukhanian P. Library construction for next-generation sequencing: Overviews and challenges. *BioTechniques*, **56**(2):61–77, 2014. doi:10.2144/000114133.

Honaas LA, Altman NS, and Krzywinski M. Study design for sequencing studies. In *Methods in Molecular Biology*, vol. 1418, pp. 39–66, 2016. doi:10.1007/978-1-4939-3578-9_3.

Hooper JE. A survey of software for genome-wide discovery of differential splicing in RNA-Seq data. *Human Genomics*, 2014. doi:10.1186/1479-7364-8-3.

Illumina. RNA-Seq Data Comparison with Gene Expression Microarrays, 2011. URL `http://www.europeanpharmaceuticalreview.com/wp-content/uploads/Illumina_whitepaper.pdf`.

Jänes J, Hu F, Lewin A, and Turro E. A comparative study of RNA-seq analysis strategies. *Briefings in Bioinformatics*, (January):1–9, 2015. doi:10.1093/bib/bbv007.

Jiang L, Schlesinger F, Davis CA, Zhang Y, Li R, Salit M, Gingeras TR, and Oliver B. Synthetic spike-in standards for RNA-seq experiments. *Genome Research*, **21**(9):1543–1551, 2011. doi:10.1101/gr.121095.111.

Kanitz A, Gypas F, Gruber AJ, Gruber AR, Martin G, and Zavolan M. Comparative assessment of methods for the computational inference of transcript isoform abundance from RNA-seq data. *Genome Biology*, **16**(1), 2015. doi:10.1186/s13059-015-0702-5.

Karimpour-Fard A, Epperson LE, and Hunter LE. A survey of computational tools for downstream analysis of proteomic and other omic datasets. *Human Genomics*, **9**(28), 2015. doi:10.1186/s40246-015-0050-2.

Katz Y, Wang ET, Airoldi EM, and Burge CB. Analysis and design of RNA sequencing experiments for identifying isoform regulation. *Nature Methods*, **7**(12):1009–1015, 2010. doi:10.1038/nmeth.1528.

Katz Y, Wang ET, Silterra J, Schwartz S, Wong B, Thorvaldsdóttir H, Robinson JT, Mesirov JP, Airoldi EM, and Burge CB. Quantitative visualization of alternative exon expression from RNA-seq data. *Bioinformatics (Oxford, England)*, 2015. doi:10.1093/bioinformatics/btv034.

Kim D, Pertea G, Trapnell C, Pimentel H, Kelley R, and Salzberg SL. TopHat2: accurate alignment of transcriptomes in the presence of insertions, deletions and gene fusions. *Genome Biology*, **14**(4):R36,

© 2015-2018 Applied Bioinformatics Core | Weill Cornell Medical College

2013. doi:10.1186/gb-2013-14-4-r36.

Kundaje A. A comprehensive collection of signal artifact blacklist regions in the human genome. Tech. rep., 2013. URL `https://sites.google.com/site/anshulkundaje/projects/blacklists`.

Law CW, Chen Y, Shi W, and Smyth GK. voom: precision weights unlock linear model analysis tools for RNA-seq read counts. *Genome Biology*, **15**:R29, 2014. doi:10.1186/gb-2014-15-2-r29.

Levin JZ, Yassour M, Adiconis X, Nusbaum C, Thompson DA, Friedman N, Gnirke A, and Regev A. Comprehensive comparative analysis of strand-specific RNA sequencing methods. *Nature Methods*, **7**(9):709–715, 2010. doi:10.1038/nmeth.1491.

Li B and Dewey C. RSEM: accurate transcript quantification from RNA-Seq data with or without a reference genome. *BMC Bioinformatics*, **12**(1):323, 2011. doi:10.1186/1471-2105-12-323.

Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N, Marth G, Abecasis G, and Durbin R. The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, **25**(16):2078–9, 2009. doi: 10.1093/bioinformatics/btp352.

Li S, Labaj PP, Zumbo P, Sykacek P, Shi W, Shi L, Phan J, Wu PY, Wang M, Wang C, Thierry-Mieg D, Thierry-Mieg J, Kreil DP, and Mason CE. Detecting and correcting systematic variation in large-scale RNA sequencing data. *Nat Biotech*, **32**(9):888–895, 2014. doi:10.1038/nbt.3000.

Liao Y, Smyth GK, and Shi W. featureCounts: an efficient general purpose program for assigning sequence reads to genomic features. *Bioinformatics*, **30**(7):923–30, 2014. doi:10.1093/bioinformatics/btt656.

Liu Y, Zhou J, and White KP. RNA-seq differential expression studies: more sequence or more replication? *Bioinformatics*, **30**(3):301–4, 2014. doi:10.1093/bioinformatics/btt688.

Love MI, Huber W, and Anders S. Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. *Genome Biology*, **15**(12):550, 2014. doi:10.1186/s13059-014-0550-8.

Lowe R, Shirley N, Bleackley M, Dolan S, and Shafee T. Transcriptomics technologies. *PLoS Computational Biology*, 2017. doi:10.1371/journal.pcbi.1005457.

Meng C, Zeleznik OA, Thallinger GG, Kuster B, Gholami AM, and Culhane AC. Dimension reduction techniques for the integrative analysis of multi-omics data. *Briefings in Bioinformatics*, **17**(4):628–641, 2016. doi:10.1093/bib/bbv108.

Moreton J, Izquierdo A, and Emes RD. Assembly, assessment and availability of de novo generated eukaryotic transcriptomes. *Frontiers in Genetics*, **6**(January):1–9, 2015. doi:10.3389/fgene.2015.00361.

Munro SA, Lund SP, Pine PS, Binder H, Clevert DA, Conesa A, Dopazo J, Fasold M, Hochreiter S, Hong H, Jafari N, Kreil DP, Łabaj PP, Li S, Liao Y, Lin SM, Meehan J, Mason CE, Santoyo-Lopez J, Setterquist RA, Shi L, Shi W, Smyth GK, Stralis-Pavese N, Su Z, Tong W, Wang C, Wang J, Xu J, Ye Z, Yang Y, Yu Y, Salit M, Labaj PP, Li S, Liao Y, Lin SM, Meehan J, Mason CE, Santoyo-Lopez J, Setterquist RA, Shi L, Shi W, Smyth GK, Stralis-Pavese N, Su Z, Tong W, Wang C, Wang J, Xu J, Ye Z, Yang Y, Yu Y, and Salit M. Assessing technical performance in differential gene expression experiments with external spike-in RNA control ratio mixtures. *Nature Communications*, **5**:5125, 2014. doi:10.1038/ncomms6125. URL `http://www.bioconductor.org/packages/release/bioc/vignettes/erccdashboard/inst/doc/erccdashboard.pdf`.

Nookaew I, Papini M, Pornputtapong N, Scalcinati G, Fagerberg L, Uhlén M, and Nielsen J. A comprehensive comparison of RNA-Seq-based transcriptome analysis from reads to differential gene expression and cross-comparison with microarrays: A case study in Saccharomyces cerevisiae. *Nucleic Acids Research*, **40**(20):10084–10097, 2012. doi:10.1093/nar/gks804.

NuGEN. Detection of Genomic DNA in Human RNA Samples for RNA-Seq, 2013. URL `http://www.nugen.com/sites/default/files/M01355_v1-TechnicalReport,DetectionofGenomicDNAinHumanRNASamplesforRNA-Seq.pdf`.

Oshlack A and Wakefield MJ. Transcript length bias in RNA-seq data confounds systems biology. *Biology Direct*, **4**:14, 2009. doi:10.1186/1745-6150-4-14.

Patro R, Duggal G, Love MI, Irizarry RA, and Kingsford C. Salmon provides fast and bias-aware quantification of transcript expression. *Nature Methods*, **14**(4):417–419, 2017. doi:10.1038/nmeth.4197.

Patro R, Mount SM, and Kingsford C. Sailfish enables alignment-free isoform quantification from RNA-seq reads using lightweight algorithms. *Nature Biotechnology*, **32**(5):462–464, 2014. doi:10.1038/nbt.2862.

Pimentel HJ. RNA-Seq Methods and Algorithms (Part III âĂŞ Quantification), . URL `https://www.youtube.com/watch?v=ztyjiCCt_lM`.

Pimentel HJ, Bray N, Puente S, Melsted P, and Pachter L. Differential analysis of RNA-Seq incorporating quantification uncertainty. *bioRxiv*, p. 058164, 2016. doi:10.1101/058164.

Rapaport F, Khanin R, Liang Y, Pirun M, Krek A, Zumbo P, Mason CE, Socci ND, and Betel D. Comprehensive evaluation of differential gene expression analysis methods for RNA-seq data. *Genome Biology*, **14**(9):R95, 2013. doi:10.1186/gb-2013-14-9-r95.

Risso D, Ngai J, Speed TP, and Dudoit S. Normalization of RNA-seq data using factor analysis of control genes or samples. *Nature Biotechnology*, pp. 1–10, 2014. doi:10.1038/nbt.2931.

Ritchie ME, Phipson B, Wu D, Hu Y, Law CW, Shi W, and Smyth GK. limma powers differential expression analyses for RNA-sequencing and microarray studies. *Nucleic Acids Research*, **43**(7):e47–, 2015. doi:10.1093/nar/gkv007.

Roberts A and Pachter L. Streaming fragment assignment for real-time analysis of sequencing experiments. *Nature Methods*, **10**(1):71–73, 2013. doi:10.1038/nmeth.2251.

Robinson MD, McCarthy DJ, and Smyth GK. edgeR: a Bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics*, **26**(1), 2010. doi:10.1093/bioinformatics/btp616.

Robinson MD and Oshlack A. A scaling normalization method for differential expression analysis of RNA-seq data. *Genome Biology*, **11**(3):R25, 2010. doi:10.1186/gb-2010-11-3-r25.

Schurch NJ, Schofield P, Gierliński M, Cole C, Sherstnev A, Singh V, Wrobel N, Gharbi K, Simpson GG, Owen-Hughes T, Blaxter M, and Barton GJ. How many biological replicates are needed in an RNA-seq experiment and which differential expression tool should you use? *RNA*, pp. 1–13, 2016. doi:10.1261/rna.053959.115.

Schurch NJ, Schofield P, Gierliński M, Cole C, Simpson GG, Hughes TO, Blaxter M, and Barton GJ. Evaluation of tools for differential gene expression analysis by RNA-seq on a 48 biological replicate experiment. *ArXiv e-prints*, 2015. URL `http://arxiv.org/abs/1505.02017`.

Seyednasrollah F, Laiho A, and Elo LL. Comparison of software packages for detecting differential expression in RNA-seq studies. *Briefings in Bioinformatics*, **16**(1):59–70, 2015. doi:10.1093/bib/bbt086.

Shanker S, Paulson A, Edenberg HJ, Peak A, Perera A, Alekseyev YO, Beckloff N, Bivens NJ, Donnelly R, Gillaspy AF, Grove D, Gu W, Jafari N, Kerley-Hamilton JS, Lyons RH, Tepper C, and Nicolet CM. Evaluation of commercially available RNA amplification kits for RNA sequencing using very low input amounts of total RNA. *Journal of Biomolecular Techniques*, 2015. doi:10.7171/jbt.15-2601-001.

Shen S, Park JW, Lu Zx, Lin L, Henry MD, Wu YN, Zhou Q, and Xing Y. rMATS: Robust and flexible detection of differential alternative splicing from replicate RNA-Seq data. *PNAS*, 2014. doi:10.1073/pnas.1419161111.

Sims D, Sudbery I, Ilott NE, Heger A, and Ponting CP. Sequencing depth and coverage: key considerations in genomic analyses. *Nature Reviews Genetics*, **15**(2):121–32, 2014. doi:10.1038/nrg3642.

Smyth GK. Linear models and empirical bayes methods for assessing differential expression in microarray experiments. *Statistical Applications in Genetics and Molecular Biology*, **3**, 2004. doi:10.2202/1544-6115.1027.

Soneson C and Delorenzi M. A comparison of methods for differential expression analysis of RNA-seq data. *BMC Bioinformatics*, **14**(1):91, 2013. doi:10.1186/1471-2105-14-91.

Soneson C, Love MI, and Robinson MD. Differential analyses for RNA-seq: transcript-level estimates improve gene-level inferences. *F1000Research*, **4**(0):1521, 2015. doi:10.12688/f1000research.7563.2.

Soneson C, Matthes KL, Nowicka M, Law CW, and Robinson MD. Isoform prefiltering improves performance of count-based methods for analysis of differential transcript usage. *Genome Biology*, **17**(1):12, 2016. doi:10.1186/s13059-015-0862-3.

Su Z, Łabaj PP, Li SS, Thierry-Mieg J, Thierry-Mieg D, Shi W, Wang C, Schroth GP, Setterquist Ra, Thompson JF, Jones WD, Xiao W, Xu W, Jensen RV, Kelly R, Xu J, Conesa A, Furlanello C, Gao HH, Hong H, Jafari N, Letovsky S, Liao Y, Lu F, Oakeley EJ, Peng Z, Praul CA, Santoyo-Lopez J, Scherer A, Shi T, Smyth GK, Staedtler F, Sykacek P, Tan XX, Thompson EA, Vandesompele J, Wang MD, Wang JJJ, Wolfinger RD, Zavadil J, Auerbach SS, Bao W, Binder H, Blomquist T, Brilliant MH, Bushel PR, Cai W, Catalano JG, Chang CW, Chen T, Chen G, Chen R, Chierici M, Chu TM, Clevert DA, Deng Y, Derti A, Devanarayan V, Dong Z, Dopazo J, Du T, Fang H, Fang Y, Fasold M, Fernandez A, Fischer M, Furió-Tari P, Fuscoe JC, Caimet F, Gaj S, Gandara J, Gao HH, Ge W, Gondo Y, Gong B, Gong M, Gong Z, Green B, Guo C, Guo LWL, Guo LWL, Hadfield J, Hellemans J, Hochreiter S, Jia M, Jian M, Johnson CD, Kay S, Kleinjans J, Lababidi S, Levy S, Li QZ, Li L, Li P, Li Y, Li H, Li J, Li SS, Lin SM, López FJ, Lu X, Luo H, Ma X, Meehan J, Megherbi DB, Mei N, Mu B, Ning B, Pandey A, Pérez-Florido J, Perkins RG, Peters R, Phan JH, Pirooznia M, Qian F, Qing T, Rainbow L, Rocca-Serra P, Sambourg L, Sansone SA, Schwartz S, Shah R, Shen J, Smith TM, Stegle O, Stralis-Pavese N, Stupka E, Suzuki Y, Szkotnicki LT, Tinning M, Tu B, van Delft J, Vela-Boza A, Venturini E, Walker SJ, Wan L, Wang W, Wang JJJ, Wang JJJ, Wieben ED, Willey JC, Wu PY, Xuan J, Yang Y, Ye Z, Yin Y, Yu Y, Yuan YC, Zhang J, Zhang KK, Zhang WW, Zhang WW, Zhang Y, Zhao C, Zheng Y, Zhou Y, Zumbo P, Tong W, Kreil DP, Mason CE, and Shi L. A comprehensive assessment of RNA-seq accuracy, reproducibility and information content by the Sequencing Quality Control Consortium. *Nat Biotech*, **32**(9):903–14, 2014. doi:10.1038/nbt.2957.

Sultan M, Amstislavskiy V, Risch T, Schuette M, Dökel S, Ralser M, Balzereit D, Lehrach H, and Yaspo ML. Influence of RNA extraction methods and library selection schemes on RNA-seq data. *BMC Genomics*, **15**(1):675, 2014. doi:10.1186/1471-2164-15-675.

't Hoen PAC, Friedländer MR, Almlöf J, Sammeth M, Pulyakhina I, Anvar SY, Laros JFJ, Buermans HPJ, Karlberg O, Brännvall M, The GEUVADIS Consortium, den Dunnen JT, van Ommen GJB, Gut IG, Guigó R, Estivill X, Syvänen AC, Dermitzakis ET, and Lappalainen T. Reproducibility of high-throughput mRNA and small RNA sequencing across laboratories. *Nat Biotech*, **31**(11):1015–1022, 2013. doi:10.1038/nbt.2702.

Teng M, Love MI, Davis CA, Djebali S, Dobin A, Graveley BR, Li S, Mason CE, Olson S, Pervouchine D, Sloan CA, Wei X, Zhan L, and Irizarry RA. A benchmark for RNA-seq quantification pipelines. *Genome Biology*, **17**(1), 2016. doi:10.1186/s13059-016-0940-1.

Trapnell C, Hendrickson DG, Sauvageau M, Goff L, Rinn JL, and Pachter L. Differential analysis of gene regulation at transcript resolution with RNA-seq. *Nat Biotech*, **31**(1):46–53, 2013. doi:10.1038/nbt.2450.

Trapnell C, Roberts A, Goff L, Pertea G, Kim D, Kelley DR, Pimentel H, Salzberg SL, Rinn JL, and Pachter L. Differential gene and transcript expression analysis of RNA-seq experiments with TopHat and Cufflinks. *Nature Protocols*, **7**(3):562–78, 2012. doi:10.1038/nprot.2012.016.

Wagner GP, Kin K, and Lynch VJ. Measurement of mRNA abundance using RNA-seq data: RPKM measure is inconsistent among samples. *Theory in Biosciences*, **131**(4):281–285, 2012. doi:10.1007/s12064-012-0162-3.

Wang L, Wang S, and Li W. RSeQC: Quality control of RNA-seq experiments. *Bioinformatics*, **28**(16):2184–2185, 2012. doi:10.1093/bioinformatics/bts356.

Williams CR, Baccarella A, Parrish JZ, and Kim CC. Empirical assessment of analysis workflows for differential expression analysis of human samples using RNA-Seq. *BMC Bioinformatics*, **18**(1), 2017. doi:10.1186/s12859-016-1457-z.

Yeri A, Courtright A, Danielson K, Hutchins E, Alsop E, Carlson E, Hsieh M, Ziegler O, Das A, Shah RV, Rozowsky J, Das S, and Van Keuren-Jensen K. Evaluation of commercially available small RNASeq library preparation kits using low input RNA. *BMC Genomics*, 2018. doi:10.1186/s12864-018-4726-6.

Yona G, Dirks W, and Rahman S. Comparing algorithms for clustering of expression data: how to assess gene clusters. *Methods Mol Biol*, **541**:479–509, 2009. doi:10.1007/978-1-59745-243-4_21.

Zeng W and Mortazavi A. Technical considerations for functional sequencing assays. *Nature Immunology*, **13**(9):802–807, 2012. doi:10.1038/ni.2407.

Zhao S, Fung-Leung WP, Bittner A, Ngo K, and Liu X. Comparison of RNA-Seq and microarray in transcriptome profiling of activated T cells. *PLoS ONE*, **9**(1), 2014. doi:10.1371/journal.pone.0078644.