Introduction

Overall, RNA-Seq analysis consists in a series of extraction, preprocessing and statistical analysis (Fig. 1):

1. DNA extraction from a sample
2. DNA sequencing
3. Alignment of sequencing reads to a reference genome
4. Basic exploratory data analysis
5. Identification of genomic variants (SNPs, small insertions and deletions)
6. Gene quantification (i.e., statistics on count data)

This tutorial starts from step 3, and we will skip step 5, which is covered in another tutorial (using `mpileup` and varscan). A typical workflow will take 4-6 hours, most of this time being specnt in file generation and exploratory data analysis. Indeed, whether data processing is done in Galaxy or at the command-line, a lot of the above steps require converting file from one format to the other, sorting data according to genomic coordiantes and creating index file to ease the processing of the data file. For instance, an SRA file has to be converted to a Fastq file and a pseudo-alignment BAM file needs to be indexed: while the later will run under 3 minutes, the former takes a much longer time. Likewise, soritng a SAM or BAM file may be quite time consuming for large files.

Some reviews of best practices are available (Conesa et al. 2016; Yendrek, Ainsworth, and Thimmapuram 2012). Regarding statistical software, the state of the art is described on the Bioconductor project (Korpelainen et al. 2015; Anders et al. 2013).

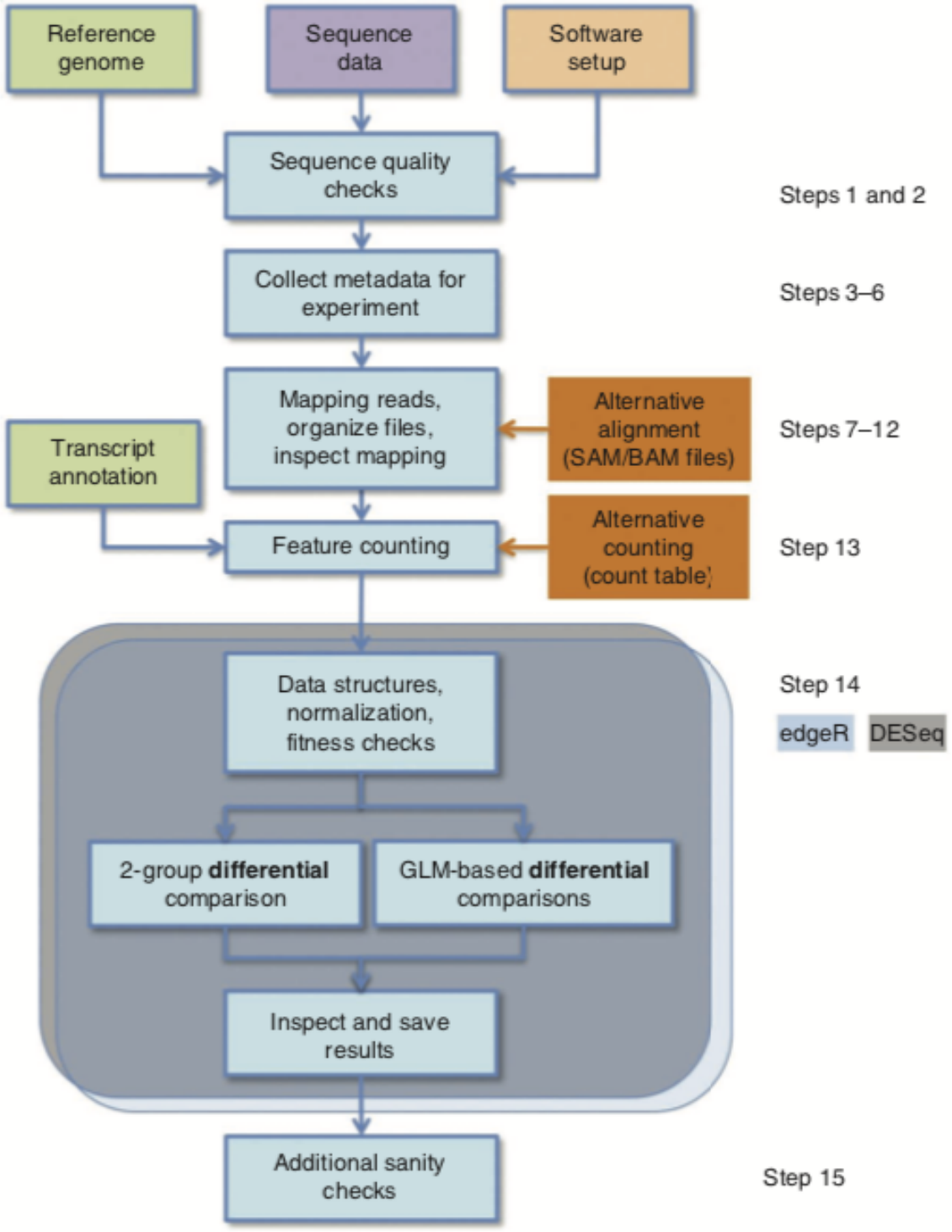
Reference genome
Sequence data
Software setup
Sequence quality checks
Steps 1 and 2
Collect metadata for experiment
Steps 3–6
Mapping reads, organize files, inspect mapping
Alternative alignment (SAM/BAM files)
Steps 7–12
Transcript annotation
Feature counting
Alternative counting (count table)
Step 13
Data structures, normalization, fitness checks
Step 14
edgeR
DESeq
2-group differential comparison
GLM-based differential comparisons
Inspect and save results
Additional sanity checks
Step 15

Figure 1

Applications

Preprocessing (read mapping)

Kallisto + DESeq2

Whether we use Galaxy or the command-line, we basically need two kind of files: a reference genome (Fasta) and Fastq or sra reads files. If reads are in SRA format, they need to be converted to Fastq format using the `sratoolkit`. For gene quantification using `DESeq2`, we will further need annotation as a GFF or GFF3 file.

Kallisto will produce counts and TPM values in a TSV file (as well as an HDF5 file). Here is an example of Kallisto output:

```
target_id       length  eff_length      est_counts      tpm
jgi|Podans1|100005|CE99640_2373 1994    1895    1407    51.6298
jgi|Podans1|100060|CE99695_1372 3146    3047    504     11.502
jgi|Podans1|10015|CE9650_8706   1582    1483    1156    54.2042
```

Note that transcript ID (taken from the reference genome, `>jgi|Podans1|100005|CE99640_2373`) should correspond to gene ID in the annotation file, which actually reads:

```
##gff-version 3
-%<-----
38763:scaffold_3    prediction  gene    644807  646915  0   +   .   ID=
   gene_5098;Name=jgi.p|Podans1|99641;portal_id=Podans1;product_name=
   Cytochrome P450;proteinId=99641;transcriptId=100005
38764:scaffold_3    prediction  mRNA    644807  646915  .   +   .   ID=
   mRNA_5098;Name=jgi.p|Podans1|99641;Parent=gene_5098;proteinId=99641;
   track=FilteredModels1;transcriptId=100005
```

In this case, this means that the transcript ID 100005 is found in unexpected place: when using Galaxy, you will have to update the corresponding ID.[1]

Basically, NGS analyses (RNA, CHIP, etc.) need to account for within-group variance estimates when analysing lot of genes, hence the need to pool information across genes. The DESeq approach detects and corrects dispersion estimates that are too low through modeling of the dependence of the dispersion on the average expression strength over all samples. In addition, it provides a novel method for gene ranking and the visualization of stable estimates of effect sizes (Love, Huber, and Anders 2014). The DESeq2 package further includes shrunken fold changes (with SE).

[1] An R script can be used to process the original GFF file and update ID so that they match each other.

We need two data frames before running the statistical analysis: a table of counts (or TPM values from `kallisto` or `salmon`), and a table describing the experimental condition. Note that each run must come as triplicate. Here are the basic steps to import counts data and setup the design matrix:

```
library("DESeq2")
library("tximport")
datadir <- dir(".", pattern = "out.\\.*")
files <- file.path(datadir, "abundance.h5")
names(files) <- gsub("out.", "", datadir)
samples <- data.frame(run = names(files), condition = c("4h", "48h", "
    24h"))
txi <- tximport(files, "kallisto", txOut = TRUE)
dds <- DESeqDataSetFromTximport(txi, colData = samples, design = ~
    condition)
```

TopHat + HTSeq (or FeatureCounts)

This workflow usually takes a longer time but TopHat has the advantage of being aware of splice junction (in fact, this is what TopHat really adds to bowtie).

Statistical analysis of count data

Sanity checking

Two preprocessing stages are worth considering before dwelling into gene quantification *per se*: quality control and normalization. Quality control amounts to controlling the number of genes having non-zero counts in all samples. If this number is very low, it is likely that something went wrong, at least with some of the samples. Based on the DESeq table of counts, we can proceed as follows in R:

```
gene_counts <- counts(dds)
counts_per_sample <- apply(gene_counts, 1, function(x) all(x) > 0)
cat(sum(counts_per_sample), "out of", nrow(dds), "\n")
```

Normalization refers to the comparison of size factors defined as the median of ratios of each sample to a virtual reference sample (median of each gene's values across samples). Those ratios are expected to match the ratios of the library sizes and be roughly equal to one. Dividing each column of the count table by the corresponding size factor yields normalized count values, which can be scaled to give a counts per million interpretation. Note that this is different from the approach taken by `edgeR`, which considers the trimmed mean of M values.

The MA plot shows the average vs mean–difference of log fold change, centered around 0, and it is expected to observe higher variability of log ratios at lower counts.

Sample clustering

A PCA or simple heatmap of the results of hierarchical clustering of the sample data can be used to assess overall similarity between samples. We expect triplicate to cluster together while samples from very different experimental conditions are expected to be far away one from the other. Of note, it is useful to apply a regularized log–transformation on the raw counts to avoid the impact of few highly variale genes, hence considering a roughly equal contribution from all genes. For genes with high counts, this mostly resembles a log2 transformation, whereas for genes with low counts, this shrunkens values toward gene's average across amples. It is important to note that this is only for exploratory analysis; for statistical modeling, raw counts should be preferred since DESeq will handle appropriate correction automagically.

```
rld <- rlogTransformation(dds, blind = TRUE)
dd <- as.matrix(dist(t(assay(rld))))
library(pheatmap)
pheatmap(dd)
plotPCA(rld, intgroup = c("condition"))
```

Differential analysis

The statistical model underlying differential analysis of count data is a Negative Binomial, which contrary to the standard Poisson model allows to account for overdispersion (i.e., variance greater than mean). Variance is modeled as $\mathbb{V}[NB(\mu, \alpha)] = \mu + \alpha\mu^2$, and the very first step in differential analysis is to get an estimate of the dispersion parameter for each gene (independent of the condition, which is sensible since there is usually a low number of replicates). Notice that for genes with very low read counts, the large amount of Poisson noise prevent those genes from exhibiting any DE at all, and DESeq performs independent filtering automatically to discard such low signals and to increase statistical power for the remaining gene candidates. The asymptotic dispersion for highly expressed genes can be seen as a measurement of biological variability in the sense of a squared coefficient of variation: a dispersion value of 0.01 means that the gene's expression tends to differ by typically $\sqrt{0.01} = 10\%$ between samples of the same treatment group. The R procedure `estimateDispersions` allows to compute (and visualize) dispersion estimates as a function of mean normalized counts.

The statistical test used to assess whether genes are differentially expressed between samples is a Wald test (`nbinomWaldTest`), with FDR correction for multiple testing. Benjamini–Hochberg's adjusted

p-values can then be ranked to highlight the top genes. Usually, the statistical threshold is set at 0.1, and not 0.05 as in standard null hypothesis statistical testing.

```
ddstab <- estimateDispersions(estimateSizeFactors(dds))
r <- results(nbinomWaldTest(ddstab), pAdjustMethod = "BH")
table(r$padj < 0.1)
## r@metadata$filterThreshold
```

The inspection of the distribution of (unadjusted) p-values is helpful to verify that the null distribution for the test statistic is viable. If the histogram does not exhibit an uniform pattern (e.g., U or hill shape), then it is likely that the $\mathcal{N}(0,1)$ null distribution is not appropriate.[2]

Once the model is fitted, it is possible to extract the top genes using, e.g.:

```
plotMA(r)
names(dds)[which(r$padj < 0.1)]
```

Gene ontology and annotation

This is beyond the scope of this tutorial, but the genefilter package can be used to send queries about specific gene sets, or it is possible to use the online DAVID package.

Appendix

File Formats

Fasta files contain nucleotide or aa, with a description line (the symbol > followed by the sequence *identifier* and other (optional) information, also called *comment*). File extension is usually `.fasta`, `.fa`, or `.faa` (amino acid) and `.fna` (nucleotides).

Most bioinformatics software allow to read Fasta file in an efficient manner. Don't try to write your own reader unless you know what you are doing. E.g., in Python, you can use the BioPython module:

```
from Bio import SeqIO

records = list(SeqIO.parse("sordariales-all.fa", 'fasta'))
```

Fastq files are like Fasta file (header = `@` followed by sequence ID) but with the corresponding quality scores on a separate line (after the + sign). Quality value ranges from 0x21 (!, lowest) to 0x7e (~, highest).

[2] See the fdrtool and locfdr packages for further strategies about controlling global or local FDR, and empirical null modeling allowing to estimate the variance of the null model (expected to be 1, per the $\mathcal{N}(0,1)$ hypothesis).

Like Fasta files, they can be compressed using `gzip`. Extension is usually `.fastq` or `.fq`. Beware of differing rules (offset coding and PHRED scores) between Sanger and Illumina Fastq format. Here is an example of the first of the four lines available in each entry of a Fastq file:

```
@K00188:208:HFLNGBBXX:3:1101:1428:1508 2:N:0:CTTGTA
```

The interesting part in the above example are `2:N:0:CTTGTA`, which stands for the member of a pair (1 or 2, for paired-end or mate-pair reads only), the filter status (Y if filtered, N otherwise), the control bits (0 when none, otherwise if they are on), and the index sequence or barcode (`CTTGTA`).

Together with SRA format (see below), this is the default format for DNA reads. Quality control on the reads can be performed using, e.g., `sickle`, which basically allows to filter reads with low quality, e.g., PHRED score < 35:[3]

```
$ sickle se -f SRR391535.fastq -t sanger -o trimmed_SRR391535.fastq -q
    35 -l 45
```

This is often one of the very first pre-processing step when working with a batch of DNA reads.

The SRA format is used for Sequence Read Archives, from NCBI. They can be converted to Fastq using `fastq-dump`:

```
$ fastq-dump --gzip file.sra
```

As an alternative, `samtools` can be used to generate a BAM file:

```
$ sam-dump SRR6960207.sra \
    | samtools view -bS - > alignments/SRR6960207.bam
```

SAM (BAM) files are composed of a header (`@` lines) and alignments of the sequence against a reference genome. BAM files are compressed version of SAM files, and you will need the `samtools` to display them on screen.

[3] The PHRED score is defined as $Q = -10 \log_{10}(p)$ where where p is the probability that the corresponding base call is incorrect. A score of 10 means a probability of 1/10 (hence, 90% accuracy for base calling), while a score of 30 means a probability of 1/1000.

Col	Field	Type	Regexp/Range	Brief description
1	QNAME	String	`[!-?A-~]{1,254}`	Query template NAME
2	FLAG	Int	$[0, 2^{16} - 1]$	bitwise FLAG
3	RNAME	String	`\*|[:rname:^*=][:rname:]*`	Reference sequence NAME[10]
4	POS	Int	$[0, 2^{31} - 1]$	1-based leftmost mapping POSition
5	MAPQ	Int	$[0, 2^{8} - 1]$	MAPping Quality
6	CIGAR	String	`\*|([0-9]+[MIDNSHPX=])+`	CIGAR string
7	RNEXT	String	`\*|=|[:rname:^*=][:rname:]*`	Reference name of the mate/next read
8	PNEXT	Int	$[0, 2^{31} - 1]$	Position of the mate/next read
9	TLEN	Int	$[-2^{31} + 1, 2^{31} - 1]$	observed Template LENgth
10	SEQ	String	`\*|[A-Za-z=.]+`	segment SEQuence
11	QUAL	String	`[!-~]+`	ASCII of Phred-scaled base QUALity+33

Figure 2

VCF stands for Variant Call(ing) Format (See 1000 Genomes Project). The format is standardized, like for SAM files:

```
#CHROM, POS, ID, REF, ALT, QUAL, FILTER, INFO.
```

A typical workflow consists in calling `mpileup` via `samtools` or `bcftools` (which now integrate the pileup option). The `mpileup` command automatically scans every position supported by an aligned read, computes all the possible genotypes supported by these reads, and then computes the probability that each of these genotypes is truly present in our sample.

```
$ bcftools mpileup -f NC_008253.fna reads_aligned.sorted.bam -o
    sim_variants.bcf
```

The `vcflib` tools allow to further manipulate VCF files (comparison, format conversion, *filtering* and subsetting, annotation, samples, ordering, variant representation, genotype manipulation, interpretation and classification of variants)

Finally, GFF (or GFF3) files, which stand for General Feature Format or Gene Finding Format, are used to provide a list of the features (CDS, gene, etc.) available on a given sequence. They have 9 mandatory TAB separated columns. GFF3 have extension `.gff3`. A typical file is shown below:

```
##gff-version 3
ctg123 . mRNA            1300 9000 . + . ID=mrna0001;Name=sonichedgehog
ctg123 . exon            1300 1500 . + . ID=exon00001;Parent=mrna0001
ctg123 . exon            1050 1500 . + . ID=exon00002;Parent=mrna0001
ctg123 . exon            3000 3902 . + . ID=exon00003;Parent=mrna0001
ctg123 . exon            5000 5500 . + . ID=exon00004;Parent=mrna0001
ctg123 . exon            7000 9000 . + . ID=exon00005;Parent=mrna0001
```

Be careful with how entites are identified. In the above case, the id is called `ID`, but Galaxy expects

`gene_id`, for example.

Tools

There are many, many tools available on the internet. They are mostly open-source and macOS-compatible. To name a few:

- sickle, for QC of reads
- cufflinks, part of the Tuxedo toolsuite (Bowtie + TopHat + cufflinks)
- Picard, to manipulate HTS files
- Bowtie2, or TopHat2 which relies on Bowtie, now HISAT2 (3), for alignments
- kallisto (faster), for fast alignment
- SAMtools (4+5)
- BCFtools (not to be confounded with bedtools)
- vcflib, to manipulate VCF files
- bedtools multicov, featureCounts (subread), HTSeq

Most tools are also available in Galaxy (e.g., Galaxy Europe, or as "shed tools" on a dedicated Galaxy server). Another software that is widely used for RNA-Seq analysis is MeV, but this a web application only, see Howe et al. (2011).

Tutorials

- Blog posts: RNA seq analysis - FeatureCounts and DESeq2 workflow; Count-Based Differential Expression Analysis of RNA-seq Data
- RNA-seq workflow: gene-level exploratory analysis and differential expression
- A simple guide to variant calling with BWA, samtools, VarScan2
- Orchestrating Single-Cell Analysis with Bioconductor
- Reference-based RNA-seq data analysis (Galaxy)
- Prokaryote TNA-Seq and Functional annotation

References

Anders, Simon, Davis J. McCarthy, Yunshun Chen, Michal Okoniewski, Gordon K. Smyth, Wolfgang Huber, and Mark D. Robinson. 2013. "Count-Based Differential Expression Analysis of Rna Sequencing Data Using R and Bioconductor." *Nature Protocols* 8 (9):1765–86.

Conesa, Ana, Pedro Madrigal, Sonia Tarazona, David Gomez-Cabrero, and Alejandra et al. Cervera. 2016. “A Survey of Best Practices for Rna-Seq Data Analysis.” *Genome Biology* 17 (13):1–19.

Howe, Eleanor A., Raktim Sinha, Daniel Schlauch, and John Quackenbush. 2011. “RNA-Seq Analysis in Mev.” *Bioinformatics* 27 (22):3209–10.

Korpelainen, Eija, Jarno Tuimala, Panu Somervuo, Mikael Huss, and Garry Wong. 2015. *RNA-Seq: Data Analysis a Practical Approach*. Taylor & Francis/CRC Press.

Love, Michael I., Wolfgang Huber, and Simon Anders. 2014. “Moderated Estimation of Fold Change and Dispersion for Rna-Seq Data with Deseq2.” *Genome Biology* 15 (550):1–21.

Yendrek, Craig R, Elizabeth A. Ainsworth, and Jyothi Thimmapuram. 2012. “The Bench Scientist’s Guide to Statistical Analysis of Rna-Seq Data.” *BMC Research Notes* 5 (506):1–10.