RNA-seq for eukaryotic organisms

Instructor: Katya L. Mack
katyamack@berkeley.edu

1. Designing a RNA-seq experiment with the intention of testing for differential expression
In an ideal RNAseq experiment, all biological replicates across conditions will be reared under the same conditions and collected at the same time/developmental stage. This is not usually possible when you are not working on a model system; however, even with these controls in place, RNA-seq data suffers from an overdispersion problem (meaning the variance > mean).

To identify whether genes are differentially expressed, the difference in expression between two conditions (or species) has be greater than the uncertainty surrounding a gene's expression level within a condition (or species). Power for this test is achieved in two ways: **coverage** and **replication**.

Higher coverage versus more replicates
Uncertainty surrounding biological variation between replicates can be reduced by increasing the number of replicates. While a high number of biological replicates (at the expense of coverage) will help you call more differentially expressed genes with programs like DESeq or EdgeR, you will likely not be able to identify differential expression in genes with lower read counts. Cost-wise, replicates are also frequently more expensive (through library preparation and time to prepare or rear new samples).

A useful web based tool to help calculate whether you should invest more in coverage or in biological replicates is **Scotty:**http://bioinformatics.bc.edu/marthlab/scotty/scotty.php

Ultimately you will need to weigh the costs of increasing coverage vs. increasing the number of replicates based on your system and the goals of your experiment. Consider the following when weighing coverage vs. the number of replicates:

Are you introducing extra biological variation into your experiment?
The following can increase variation between replicates:
1. Sampling animals in the field where age and other biological factors cannot be closely controlled
2. Sampling animals of disparate ages or developmental times
3. Sampling animals reared in different facilities
4. Sampling tissues that are of heterogenous cellular composition (i.e., testis, brain, etc.)
...Increase the number of replicates!

Are you interested in:
1) Specific genes that may have low expression levels in your tissue of interest
2) The expression of specific gene families or paralogs

3) Alleles in a heterozygous individuals
...Increase coverage!

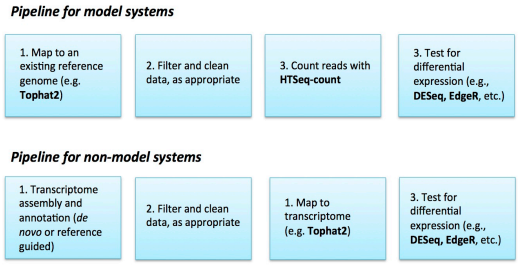
Considering Tissue Type

Gene expression is highly tissue specific. Expression is more conserved across species for a given tissue type than expression is between different tissues. This makes selecting the appropriate tissue type for your study essential. However, this can pose a problem for hypothesis driven research in systems where tissue is difficult to obtain and invasive sampling is necessary. Additionally, RNA degrades rapidly, so invasive tissue collecting needs to happen shortly after the death of an animal. It's important to consider what tissue you can get access to and how fresh it will be.

Beware of Lane Effects

Lane effects (i.e., weird biases or sequencing problems specific to a lane) can introduce serious biases into RNAseq analyses. The easy way to control for lane effects is not to distribute your treatments onto different lanes, but instead split samples across lanes. This is will result in the same coverage per sample without artificially inflating differences between conditions/treatments due to lane effects.

Overview of the pipeline



Whatever species you are working on, you ultimately need two things: 1) **a reference transcriptome** (bowtie2 or bwa) **or genome** (tophat2, hisat2, STARR) to map your reads to, and 2) a transcriptome or genome **annotation**. If you are working on a model system, these resources will be available for you to download. If you are working on a non-model system without a reference (or a closely related reference) you will need to create these tools yourself. The more closely related your species of interest is to a "model" system, the easier it will be to create these resources and the more reliable they are likely to be.

Note on filtering/cleaning RNAseq data:

How you clean your RNAseq data will depend on what platform you use and the quality of the data that is returned to you. You should spend a little bit of time looking at the FastQC files that you receive with your data to make decisions about the best way to clean your reads. Programs like trimmomatic (http://www.usadellab.org/cms/?page=trimmomatic), cutadapt (https://pypi.python.org/pypi/cutadapt), or the fastx toolkit (http://hannonlab.cshl.edu/fastx_toolkit/) can all trim reads based on quality or location (i.e., the first or last 3 bases) and remove adaptor contamination.

2. Mapping to a genome or transcriptome with Tophat2

Today we'll be using Tophat2 to map reads. There are other options for mapping your RNAseq reads (i.e., STAR, Kallisto, etc.) that may be better suited for your specific project or your computational limitations

(memory/speed).

http://ccb.jhu.edu/software/tophat

TopHat2 is a splice junction mapper for RNA-Seq reads. Tophat2 uses the short-read aligner Bowtie and analyzes the resulting mapped reads to uncover splice junctions between exons. To map reads with Tophat2 you will need a genome to map to, but you will not need a reference annotation (however, having a reference annotation will make the program run substantially faster!).

To run Tophat2, you will need **bowtie** (1 or 2) and **samtools** installed and in your path (Note: to temporary add a program to you path, type "export PATH=$PATH:/path/to/the/directory" in your terminal window. These are already added to your path in the virtualbox.)

The input for Tophat2 are FASTQ files. Both unpaired and paired end data can be mapped with Tophat, though it is recommended that you run paired and unpaired data separately.

To start, you will need to build an index for your reference genome or transcriptome. The command **bowtie2-build** builds an index file from your reference that Tophat2 and bowtie2 use to efficiently map reads.
If your reference is a fasta file, you can input the following command to build your index:

bowtie2-build

Where is the path to your reference genome or transcriptome fasta file and is what you want the prefix of your output index files to be.

As an example, we will be creating an index for an *E. coli* strain. Type the following command to index the genome of the an *E. coli* strain:
bowtie2-build Ecoli_sampledata/ecoli_genome.fa ecoli_index/ecoli_bw2

After bowtie build has finished running, you should have index files in your index folder.

Once you've built the index, you are ready to map your reads. The basic usage for Tophat2 is:

tophat [options]* PE_reads_1.fq.gz,SE_reads.fa PE_reads_2.fq.gz

Where genome_index_base is the basename for the reference followed by your reads. Reads should be in fastq format. We will test this with reads from an *E. coli* dataset (Ecoli_sampledata/SRR1783109.fastq) and the index we just built. These reads are from an experiment on *E. coli* expression under multiple oxygen conditions.

tophat ecoli_index/ecoli_bw2 Ecoli_sampledata/SRR1783109.fastq

You should see a new directory called tophat_out. In this directory is the output of our tophat run. The file "accepted_hits.bam" are the aligned reads. To check the mapping quality, click on align_summary.txt. In here, we see that 84.4% of our input mapped to the genome (not bad!). The other files in this folder

include splice juntions, insertions, and deletions that tophat identified. We will ignore these for now but know that these files are of use when looking for alternative splicing.

Pro-tip: We just ran tophat on the default settings. Some additional settings to consider when working with your own data:
--b2-sensitive or --b2-very-sensitive : Tophat tells bowtie2 to map either sensitively or very sensitively – this is slower than the default mode but will usually return more hits.
--no-discordant : Only return concordant paired-end reads
--no-mixed : Only map paired-end reads if both reads can be mapped. By default, tophat will sometimes drop one half of a paired-end read but retain the other.
--resume : Was you run terminated prematurely? Resume it instead of starting all over.
--read-mismatches : The default number of mismatches is 2. Depending on the the divergence you expect from the reference, you may want to increase this number.
-p : Set the number of threads you are working with.

3. Using samtools to manipulate BAM files

SAM (Sequence Alignment/Map) is a format for storing alignment data. Tophat2 will output data as either a SAM file or a BAM file (the compressed version of a SAM file) depending on your input settings. Our tophat output is in BAM format.

Samtools is a command line utility for manipulating BAM an SAM files. Samtools is all sorts of great applications (mpileup for creating pileup files to find variants, for one), but for now we only need to know how to convert between BAM and SAM formats and sort files to prepare for our next step (counting reads). To sort a BAM file by chromosomal coordinates, you can type:
samtools sort

To sort a file by read names rather than coordinates:
samtools sort -n

Let's sort our bam files now by read names:
samtools sort –n tophat_out/accepted_hits.bam ecoli_sortn

As SAM files are human readable and BAM files are not, it is often necessary to convert between them. You can do this with the following commands:

samtools view -h -o outputfile.sam inputfile.bam
samtools -b -S inputfile.sam > outputfile.bam

Let's do this now with our bam file:
samtools view -h -o ecoli_sortn.sam ecoli_sortn.bam

At this juncture you may want to filter your sam file for low quality alignments or reads with too many mismatches. Decisions about how to filter the file will depend on factors like the quality of your

sequencing data, length of reads, etc.

The SAM flag in column 2 of your sam file can provide a great deal of information about a given read's alignment (i.e., if the alignment unique? Did both pairs of a paired-end read map? Is this the forward or reserve strand?). Unfortunately sam flags are not written in plain English and require a computer to translate (see https://broadinstitute.github.io/picard/explain-flags.html).

Pro-tip: If you want to filter based on the number of mismatches or for uniquely mapped reads, try these handy command line one-liners on your sam file:
Retrieve only uniquely mapped reads:
awk -F'\t' '{if(($1 ~ /^@/) || ($20 ~ /NH:i:1$/)){print}}' YOUR_FILE.sam > YOUR_OUTPUTFILE.sam
Retrieve only reads with 1 or fewer mismatches:
awk -F'\t' '{if(($1 ~ /^@/) || ($17 ~ /NM:i:[0-1]$/)){print}}' YOUR_FILE.sam > YOUR_OUTPUTFILE.sam
Super pro-tip: You can adjust the above commands to filter on any column in a sam file by changing the column number (indicated by the $20 & $17 in the previous two commands) and what you are filtering on.

4. Counting reads

This step is unnecessary if you map your reads to a transcriptome.

If you have aligned your reads to a genome, you will need to count the reads mapping to each exon. An easy way to do this is to use HTSEQ-count:

http://www-huber.embl.de/users/anders/HTSeq/doc/count.html#count

HTSeq-count requires a GTF/GFF (genome annotation file) with annotated exons to count reads. As long as you have the coordinates for your exons, a GTF file will be easy to create yourself (click here to see the file specification: http://www.ensembl.org/info/website/upload/gff.html)

A gene, as counted by HTSeq, is a union of all of its exons (if you are interested in alternative splicing, exons can instead be counted as individual features, however, for identify differential expression between replicates we will be using a sum of the reads mapping to each exon associated with a given gene).

When HTSeq is installed, you can run the following command to count reads:

python -m HTSeq.scripts.count --stranded=no

If you have paired-end data, your bam files will need to be sorted (the default for HTSeq is by name, which you learned how to do in the previous section).

HTSeq-count has three running modes (option denoted as –m or **--mode=**, options are: union, intersection_strict, and intersection_nonempty) that determine how it handles read data. Which mode you choose depends on how conservative you want to be and the quality of your genome annotation. "Union" is the default mode and is applicable to most datasets.

We will not try this here with our *E. coli* dataset in the interest of time. As an example, however, there is an ecoli .gtf file in Ecoli_sampledata/ folder called ecoli_APEC.2.gtf. This is what a typical gft file looks like. This gft file and the sorted sam input file you created are all you need to count reads with HTSeq-count.

You will repeat this step for all you samples and then merge these individual files into one table (hint: if you have mapped all your reads to the same genome, use the "paste" command line tool to combine columns (i.e., on the command line: paste -d"\t" countfile_1 countfile_2….). See Mus_sampledata/Sample_dataset_Mus.txt for an output example. Alternatively use this python script (instructions at the top of the file):

merge_tables (1).py

- Details
- Download
- 2 KB

Once we have a text file with read counts for all our genes, its time to call differential expression!

---

5. Choosing how you call differential expression

Choosing how to test for differential expression not only depends on the purpose of your experiment, but also the kind of data you have collected for your analysis. RNAseq data suffers from an overdispersion problem, meaning that it has substantial variability and it cannot usually be modeled as a Poisson distribution. Programs designed for RNAseq analysis attempt to model dispersion to find genes that are differentially expressed between treatments. DESeq and EdgeR are two of the most popular programs and use slightly different statistics to call differential expression. Both of these programs are R packages that are easy to install and run and take in the count tables you learned to create in the previous step.

DESeq – DESeq models the variance in counts using a negative binomial distribution. DESeq performs best with multiple replicates. The primary change from DESeq to DESeq2 is how the program handles outliers. DESeq2 uses a test called cook's distance to remove outliers, which reduces the number of genes tested and the effects of a FDR correction.The downside is that some of these outliers may actually be differentially expressed and will not be analyzed with this approach.
http://bioconductor.org/packages/release/bioc/html/DESeq.html AND ***
https://bioconductor.org/packages/release/bioc/html/DESeq2.html***

EdgeR – EdgeR also models variance based on a negative binomial distribution. The main difference

between DESeq and EdgeR is how they handle normalization and model dispersion. Where DESeq normalizes based on a hypergeometric mean, by default EdgeR normalizes based on a trimmed mean. Practically, this translates to EdgeR frequently being more permissive in what qualifies as differentially expressed, especially for low read count genes. Differences between the results of DESeq and EdgeR are greatest when there is no biological replication.
http://www.bioconductor.org/packages/release/bioc/html/edgeR.html

Further reading:
Dillies et al. 2014. A comprehensive evaluation of normalization methods for Illumina high-throughput RNA sequencing data analysis.
Rapaport et al. 2014. Comprehensive evaluation of differential gene expression analysis methods for RNA-seq data.
Seyednasrollah et al. 2015. Comparison of software packages for detecting differential expression in RNA-seq studies.
Soneson and Delorenzi 2013. A comparison of methods for differential expression analysis of RNA-seq data.
Zhang et al. 2014. A Comparative Study of Techniques for Differential Expression Analysis on RNA-Seq Data.

6. Using DESeq2 to identify differentially expressed genes

In this example we are going to test for differential expression between light mice and dark mice in the hopes of finding differentially expressed genes related to this color polymorphism. Change directories to look at the sample data:
cd Mus_sampledata/
head Sample_dataset_Mus.txt

Open R (you can just type “R” on the command line if you have this program installed) and then load the DESeq2 library:

```
library(DESeq2)
```

Read in your table of count data, which we have named “Sample_dataset_Mus.txt”:
countdata