Guide to analyzing RNA-seq data

From BioinfWiki my Matthew D. Young

Contents

Introduction

This guide is meant to offer an easy to follow guide to the analysis of RNA-seq data, aimed at those without any prior experience analyzing next-gen data. However, a basic level of familiarity with R, the next-gen sequencing procedures and using the unix shell are assumed. It was primarily written by Matthew Young (myoung@wehi.edu.au) and is a work in progress. Most of the steps described here are outlined in our review article [5] which can be cited if people are using this guide in their work...

What does the data look like?

For every sample on which RNA-seq is run, the output you will receive is a file containing millions of short (25-300bp) DNA sequences, called reads, and quality scores indicating the confidence of each base call. However, there are some important common variations on this which depend on the platform and protocol used. These include, but are not limited to:

Base space or Colour Space

Paired end/mate pair or Single end/unpaired

Stranded or Unstranded

Base space vs Colour space

The two main platforms for second-generation sequencing of RNA are produced by Illumina and ABI Solid. While both produce millions of short reads, they are sequenced and reported in slightly different ways. The Solid platform uses a sequencing technique which generates the read information by attaching two base pairs at a time. Each base pair of a read is sequenced twice, by two (potentially different) di-nucleotides. To take advantage of this sequencing chemistry, Solid reports its reads not as a sequence of nucleotides, but as a sequence of 4 colours, where each colour represents a transition between bases, known as "colour space encoding" (http://marketing.appliedbiosystems.com/images/Product_Microsites/Solid_Knowledge_MS/pdf/SOLiD_Dibase_Sequencing_and_Color_Space_Analysis.pdf) . On the other hand, the Illumina platform reads one base at a time along a fragment until the desired read length is reached. The observed bases are then reported as the output.

The consequence of this is that the tools for analyzing RNA-seq data depend on the platform used to produce the short read data. While it is possible to convert colour space reads to base space (and visa versa), doing so introduces severe biases into the data which should be avoided at all costs. Therefore, reads should be kept in their native format and the appropriate tools should be used to analyze them.

Paired end/Mate pair reads

The standard RNA-seq protocol involves random shearing of reverse transcribed mRNA (cDNA), followed be sequencing a short "read" from one end of the fragment. This means that only the first 25-300bp of a fragment are known (depending on the length of the reads) with the rest of the fragment remaining unsequenced. In fact, because fragmentation is random, the length of each fragment is also unknown, although a size selection step is usually applied.

Although the chemistry is not sufficiently precise to allow the entire fragment to be sequenced, a clever trick can be applied whereby a short read is taken from both ends of the fragment resulting in a pair of short reads one from each end of the fragment. Reads where this has been performed are known as paired end or mate pair reads. Paired end reads allow additional information to be inferred about the intervening sequence and are particularly useful for de novo transcriptome construction and detecting structural variants, such as indels.

Stranded reads

RNA-seq data can come in either stranded or unstranded varieties. If the data is unstranded, the strand from which the fragment was transcribed cannot be identified directly from the sequence. Furthermore, because the RNA-seq protocol usually involves forming double stranded cDNA for ease of sequencing, the returned sequence is just as likely to be that of the reverse complement of the source DNA sequence as the original DNA sequence. In practical terms, this means that while half the reads map to the forward strand and half the reverse, this mapping does not contain any information about which strand the RNA was transcribed from.

Stranded RNA-seq data on the other hand preserves strand information, making it possible to identify which strand the RNA was transcribed from.

Quality Control

There are many possible avenues for performing checks for quality on sequencing data. Some popular options are: fastQC (http://www.bioinformatics.bbsrc.ac.uk/projects/fastqc/) , DNAA (http://seqanswers.com/wiki/DNAA)

Although more complex quality metrics can be used, a basic check that the sequence composition doesn't vary too greatly along the length of the reads and the quality scores do not dip too low are a good place to start. These checks (as well as several others) can be performed by loading the fastq files into the fastQC program.

Mapping the reads

Unless your aim is to do de novo transcriptome assembly, the first step of any analysis will be to align your millions of short reads to a reference of some kind. This is usually the reference genome for the species from which the RNA was extracted. There are many different aligners for performing short-read alignment to a reference (lists are available here (http://en.wikipedia.org/wiki/List_of_sequence_alignment_software) and here (http://seqanswers.com/wiki/Special:BrowseData/Bioinformatics%20application?Bioinformatics_method=Mapping)). Each has specific advantages and disadvantages, usually involving a trade-off between speed and sensitivity. For the purposes of this guide we will use the aligner BOWTIE because it is actively developed, fast, widely used and supports both base space and colour space reads.

As a starting point, I will assume you have the following files:

For single end reads:

sample.fa or sample.cfa (cfa for colour space, fa for base space)

For paired end reads:

sample_pair1.fa and sample_pair2.fa or sample_pair1.cfa and sample_pair2.cfa (cfa for colour space, fa for base space)

Obtaining the reference

In order to map the short reads to a reference genome, that genome has to be turned into an index which can be used by BOWTIE. A number of prebuilt indexes for common genomes can be downloaded from the BOWTIE website (http://bowtie-bio.sourceforge.net/index.shtml) . If your genome is not available you will have to construct the index yourself from a fasta file containing your reference genome, these can be obtained from UCSC (ftp://hgdownload.cse.ucsc.edu/goldenPath/) . This is accomplished using the command

```
bowtie-build reference.fa reference_name
```

The argument "reference_name" is a unique identifier that will be used to refer to this reference genome from now on. If your reads are colour space, construct a colour space index by adding the -C command:

```
bowtie-build -C reference.fa reference_name
```

This command will output 6 files named reference_name.1.ebwt, reference_name.2.ebwt, reference_name.3.ebwt, reference_name.4.ebwt, reference_name.rev.1.ebwt, and reference_name.rev.2.ebwt.

Irrespective of how you obtain the index, in order for BOWTIE to use it, the six files mentioned above need to be placed in the BOWTIE indexes directory. If you're not sure what your indexes directory is, it is pointed to by the environmental variable BOWTIE_INDEXES, so:

```
echo $BOWTIE_INDEXES
```

will display the path where you should put the index files.

If you wish to use an aligner other than BOWTIE, you will also have to build an index from the reference genome. Refer to the documentation for your preferred aligner for more information.

Aligning reads to the reference

Having constructed the reference into an index that BOWTIE can use, we now want to align our data to this index. BOWTIE offers a wealth of command line options that can be used to adjust the alignment algorithm and how it handles input/output. These command line options are described in detail in the BOWTIE manual (http://bowtie-bio.sourceforge.net/manual.shtml) , but there are a few flags that are commonly used in the analysis of RNA-seq and bear mentioning here.

The --sam or -S tells BOWTIE to output the results of the alignment in SAM format instead of the BOWTIE format. SAM (http://samtools.sourceforge.net/) is rapidly becoming the standard for reporting short read alignment and is supported by a wide range of downstream analysis tools. Unless you have a very good reason not to, this flag should always be specified.

The --best flag tells BOWTIE to guarantee that the alignment that it reports has the fewest number of mismatches to the reference out of all matches found. It also performs a few other desirable functions such as removing strand bias (see BOWTIE manual (http://bowtie-bio.sourceforge.net/manual.shtml#the-bowtie-aligner)). The trade-off for these benefits is that --best is slightly slower, but this difference is negligible in almost all instances. Note that --best does not apply to paired-end reads. This flag should be enabled unless speed is a major consideration.

Each base has a quality score associated with it which is reported on the PHRED scale (http://en.wikipedia.org/wiki/Phred_quality_score) where lower scores mean less confidence in the accuracy of the base call. For each candidate alignment, BOWTIE adds up the quality scores at bases which don't match the reference. Any match location that has a sum of mismatch quality scores greater than -e is deemed invalid and not reported. The default value of -e of 70 was optimized when read lengths tended to be shorter (~30bp), but is not appropriate for longer read lengths commonly used today. Furthermore, any true biological variation from the reference (such as SNPs) will in theory have a high quality score. Therefore, reads with SNPs will score very poorly on the sum of mismatching quality scores metric. For all these reasons, it is advised that the user increase -e beyond the default unless the reference is known to be an excellent representation of the biological source of RNA and the number of errors in the read is small.

The -p flag sets the number of simultaneous threads to use. As short read alignment is effectively infinitely parallelizable, set this to the number of CPU cores available.

With these considerations in mind, we map our short read data to our reference using the command

```
bowtie -p 8 --sam --best reference_name sample.fa aligned_reads.sam
```

If using colour space, the -C flag needs to be added.

```
bowtie -p 8 --sam -C --best reference_name sample.cfa aligned_reads.sam
```

If your data is paired end the two paired files need to be specified using the -1 and -2 flags

```
bowtie -p 8 --sam reference_name -1 sample_pair1.fa -2 sample_pair2.fa aligned_reads.sam
```

More complex alignment

The fragments of cDNA that are being sequenced originate not from the genome, but from the transcriptome. The transcriptome is formed by combining exons from the genome, which is why mapping to the genome is a good approximation. However, in doing so the ability to map any read that crosses exon-exon boundaries is lost. The longer the reads, the more this becomes an issue as a read is more likely to cross a boundary and be rendered unmappable. Depending on the desired downstream analysis, this lost coverage at exon boundaries may or may not be a problem.

Exon Junction libraries

It is possible to build exon junction libraries from known annotated exons and then try and map those reads that fail to map against this sequence. Such an approach is only capable of capturing reads that span known annotations, limiting its utility and biasing the results towards well annotated genes/genomes.

In order to do this, a new reference needs to be constructed that contains both the reference genome, as well as the new exon-junction sequence. It is important that the original genomic reference still be included in the reference, even if you are only mapping those reads that failed to align to the genome, so as the reads can compete amongst all possible mapping locations when determining alignment. You also need to decide how much sequence to take from either side of the exon-exon boundary. The amount of sequence from each exon must be less than the read length, otherwise any read which falls near an exon boundary, but not over it, will map to two locations, the exon junction library and the genome itself. This means that if you intend to trim your reads, the amount of sequence you take from each side of the exon junction must me less than the length of the trimmed read. Finally, you need to decide what combinations of exon-exon joinings you are going to consider. Do you only consider splicing within genes, within chromosomes, without exon reorientation? All of these things occur with different frequencies in different samples and the more possibilities you consider the greater the computational complexity.

To construct the junction library you first need a genome annotation. These can be downloaded from the UCSC table browser [1] (http://genome.ucsc.edu/cgi-bin/hgTables) . You can then either write your own tools to create a fasta file with all the exon junctions, or use an existing tool, such as the "Make Splice Junction Fasta" application included in the USeq software package [2] (http://useq.sourceforge.net/applications.html) . Once you have a fasta file containing your exon-junction libarary you need to combine it with the fasta file for your reference genome.

```
cat reference.fa junctions.fa >reference_and_junctions.fa
```

Then build a new bowtie index as descibed above

```
bowtie-build reference_and_junctions.fa junction_lib
```

Finally, you map the reads in the same way as to the reference genome. For example,

```
bowtie -p 8 --sam --best junction_lib sample.fa aligned_reads.sam
```

Further options

If this approach still does not map a reasonable number of reads, there are other alternative approaches that can be explored, at even greater computation cost. For example, you may try to estimate splice junctions from the data itself, using "De-novo" splice junction finders. There are many tools that attempt to do this, some examples include TopHat (http://tophat.cbcb.umd.edu/) , SplitSeek (http://solidsoftwaretools.com/gf/project/splitseek/) , PerM (http://code.google.com/p/perm/) , soapAls (http://soap.genomics.org.cn/soapals.html) . Full de novo assembly of the transcriptome is also a possibility, although very high coverage is required for this to work well. Paired end data is of huge benefit to this task as either end of a fragment mapping to a different exon is very strong evidence for a splice junction. A list of such tools can be found in table 1 of the review article here [Genome Biology review].

Differential Expression

A common use of expression assays is to look for differences in expression levels of genes or other objects of interest between two experimental conditions, such as a wildtype vs knockout. In order to do this we need to transform the data from a list of reads mapping to genomic coordinates into a table of counts. The strategy we employ here is to load the short reads into R using the Rsamtools package and then count the number of reads overlapping some annotation object, which is usually something like a collection of genes downloaded from the UCSC. Once transformed, a test can be performed to look for statistically significant differences in expression level.

Summarization of reads

Compressing aligned reads using SAMtools

In order to be able to load millions of aligned reads into memory in R, we need to create a binary compressed version of our human readable SAM output. This file will contain all the same information, but have a much smaller memory footprint as well as being quickly searchable. To create such files, we first need to install SAMtools (http://samtools.sourceforge.net/) . Next we need to construct an index of the reference, using the fasta file. This is done by executing:

```
samtools faidx reference.fa
```

Which creates a file reference.fa.fai. Next we convert SAM to BAM. BAM files contain all the same information as SAM files, but are compressed to be more space efficient and searchable.

```
samtools import reference.fa.fai aligned_reads.sam aligned_reads.bam
```

Finally we need to create an index of the reads so they can be quickly searched. In order to do this we first need to sort the BAM file.

```
samtools sort aligned_reads.bam aligned_reads_sorted
```

This will create aligned_reads_sorted.bam, which we now index.

```
samtools index aligned_reads_sorted.bam
```

Which creates the index file aligned_reads_sorted.bam.bai.

If you don't have the original fasta file for the reference because you downloaded a prebuilt index from the BOWTIE website (or because you lost the fasta file after making your own), you can rebuild the source fasta file by running the following.

```
bowtie-inspect reference_name>reference.fa
```

Working with BAM files in R

Our goal is to use R to summarize the reads by genes for each sample. To this end, we will use our newly created compressed representation of the short reads (the sorted, indexed, BAM file).

Fetching gene information

The first thing we need to do is to define the location of genes in chromosome coordinates. To do this, we use the GenomicFeatures (http://www.bioconductor.org/packages/2.6/bioc/html/GenomicFeatures.html) package. This package allows us to download gene information from the UCSC genome browser using the following commands:

```
library(GenomicFeatures)
txdb=makeTranscriptDbFromUCSC(genome='hg19',tablename='ensGene')
```

Various genomes and gene IDs are available, but as an example we will use the latest human genome and ENSEMBL gene IDs. The variable "txdb" now contains all the information we need, but in order to do anything with it we need to do some processing.

```
tx_by_gene=transcriptsBy(txdb,'gene')
```

This produces a GRangesList object which is a list of GRanges objects, where each GRanges object is a gene and the entries are the genomic coordinates of its transcripts. We are going to work out which reads overlap which genes using the countOverlaps function to overlap this object with the object containing the short reads.

Although we are choosing to summarize by including all reads that fall within a gene here, the procedure will work the same for any GRanges or GRangesList object. For example, if you wished to only include reads that overlap exons, you could create a different GRangesList object.

```
ex_by_gene=exonsBy(txdb,'gene')
```

Summarizing reads in R

First we load the aligned_reads_sorted.bam file into R.

```
library(Rsamtools)
reads=readBamGappedAlignments("aligned_reads_sorted.bam")
```

Checking compatibility of annotations and reads

Before we try and compare the reads to the annotation, we first need to do a few checks to make sure that everything will work OK. If your RNA-seq data did not come with strand information, than we cannot know which strand the read was transcribed from. However, the mapping process will map it to one strand or the other (in theory, both are equally likely), thus the reads will have a strand artificially allocated to them. When we count the number of reads overlaping a gene (or other feature), only those reads that map to the strand the gene is on will count and roughly half our reads will be lost. To avoid this we need to set the reads strand value to "*" (unknown).

Furthermore, it will often be the case that the chromosome names used by the alignment software (which are ultimately determined by the chromosome names in the fasta file for the reference genome) will differ from those given in the annotation. In order for the comparison function to work, these names need to be converted to the same naming convention. It is easy to check if the names match by listing all the chromosomes for both reads and annotations. Remember, our annotation data is stored in "tx_by_gene" and our short reads are stored in "reads".

```
#The annotations have chromosomes called
names(seqlengths(tx_by_gene))
#The reads have chromosomes called
as.character(unique(rname(reads)))
```

If the chromosome names are the same (or are the same for the ones you care about), then no name conversion is needed. If on the other hand they differ, we need to change either the reads or the annotations naming convention. It turns out that it is usually easier to change the names of the reads, but the procedure is the same regardless. This is best illustrated by an example. Suppose you have this situation

```
#The annotations have chromosomes called
> names(seqlengths(tx_by_gene))
 [1] "chr1"         "chr10"        "chr11"        "chr12"        "chr13"
 [6] "chr13_random" "chr14"        "chr15"        "chr16"        "chr17"
[11] "chr17_random" "chr18"        "chr19"        "chr1_random"  "chr2"
[16] "chr3"         "chr3_random"  "chr4"         "chr4_random"  "chr5"
[21] "chr5_random"  "chr6"         "chr7"         "chr7_random"  "chr8"
[26] "chr8_random"  "chr9"         "chr9_random"  "chrM"         "chrUn_random"
[31] "chrX"         "chrX_random"  "chrY"         "chrY_random"
```

```
#The reads have chromosomes called
>as.character(unique(rname(reads)))
 [1] "10.1-129993255" "11.1-121843856" "1.1-197195432"  "12.1-121257530"
 [5] "13.1-120284312" "14.1-125194864" "15.1-103494974" "16.1-98319150"
 [9] "17.1-95272651"  "18.1-90772031"  "19.1-61342430"  "2.1-181748087"
[13] "3.1-159599783"  "4.1-155630120"  "5.1-152537259"  "6.1-149517037"
[17] "7.1-152524553"  "8.1-131738871"  "9.1-124076172"  "MT.1-16299"
[21] "X.1-166650296"  "Y.1-15902555"
```

So we need to convert the read chromosome names "NO.1-Length" to the annotation name "chrNO".

```
new_read_chr_names=gsub("(.*)[T]*\\..*","chr\\1",rname(reads))
```

If you are not familiar with regular expressions, refer to the help file for gsub in R. new_read_chr_names will now contain the read chromosome names, converted to the same format as the annotation object tx_by_genes.

Now we can fix both the chromosome name and strand problem simultaneously, by building a GenomicRanges object from each of the read objects. If you have unstranded RNA-seq data and need to convert chromosomes, we run:

```
reads=GRanges(seqnames=new_read_chr_names,ranges=IRanges(start=start(reads),end=end(reads)),
    strand=rep("*",length(reads)))
```

If we just want to convert chromosome names.

```
reads=GRanges(seqnames=new_read_chr_names,ranges=IRanges(start=start(reads),end=end(reads)),
    strand=strand(reads))
```

If we just want to make each read be ambiguous with respect to strand.

```
reads=GRanges(seqnames=rname(reads),ranges=IRanges(start=start(reads),end=end(reads)),
    strand=rep("*",length(reads)))
```

Note that if you mapped reads to an exon-junction library, every exon junction will have its own "chromosome". If you wish these junctions reads to be included in the summarization, you will have to convert each of them to genomic coordinates. As each read comes from two distinct genomic locations (either side of the exon junction), you will have to make a decision about how you are going to assign each read a genomic coordinate.

Counting the number of reads

Finally, we get the number of reads that overlap with each gene (or whatever else you're interested in).

```
counts=countOverlaps(tx_by_gene,reads)
```

"counts" will now contain a numeric vector, where the ith entry is the number of reads that overlap the ith gene in "tx_by_gene".

Differential Expression Testing

As our aim is to compare conditions, we will have more than one lane of reads, possibly several for each condition. Using the procedure outlined in the previous section, we can count the number of reads that overlap a feature of interest, such as genes, in each experimental condition, for each replicate. Next we combine them into a table of counts.

```
toc=data.frame(condition1_rep1=counts1.1,condition1_rep2=counts1.2,
    condition2_rep1=counts2.1,coundtion2_rep2=counts2.2,stringsAsFactors=FALSE)
rownames(toc)=names(tx_by_gene)
```

etc. for as many conditions and replicates as are available. Here the convention is countsn.m is the vector containing the number of reads from replicate m of condition n that overlap the genes given by tx_by_gene.

Normalization

It has been shown that a small number of highly expressed genes can consume a significant amount of the total sequence. As this can change between lanes and experimental condition, along with library size, it is necessary to perform some kind of between sample normalization when testing for differential expression. The choice of normalization is not independent of the test used to determine if any genes are significantly differentially expressed (DE) between conditions. For example, quantile normalization produces non-integer counts, making tests based on the assumption of count data such as the widely used Poisson or Negative Binomial models inapplicable. We choose to use the scaling factor normalization method as it preserves the count nature of the data and has been shown to be an effective means of improving DE detection [1].

To perform the normalization and the test for differential expression, we will use the R package edgeR (http://www.bioconductor.org/packages/2.6/bioc/html/edgeR.html) [2], although there are other options available. We can now calculate the normalization factors using the TMM method [1]

```
library(edgeR)
norm_factors=calcNormFactors(as.matrix(toc))
```

Statistical testing

Next, we have to create a DGE object used by edgeR. The scaling factor calculated using the TMM method is incorporated into the statistical test by weighting the library sizes by the normalization factors (which are then used as an offset in the statistical model).

```
DGE=DGEList(toc,lib.size=norm_factors*colSums(toc),group=rep(c("Condition1","Condition2"),c(2,2)))
```

The group variable identifies which columns in the table of contents come from which experimental condition or "group". To perform the statistical test for significance, we first estimate the common dispersion parameter

```
disp=estimateCommonDisp(DGE)
```

Finally, we calculate the p-values for genes being DE

```
tested=exactTest(disp)
```

Gene Set testing (GO)

To accurately test sets of genes for over representation amongst DE genes using RNA-seq data, we need to use a method which takes into account the biases particular to this technology. The goseq package (http://www.bioconductor.org/packages/2.6/bioc/html/goseq.html) is one such method for accounting for certain RNA-seq specific biases when performing GO (and other gene set based tests) analysis [3]. First we must format the output of edgeR to be read by goseq. We call any gene with a Benjamini-Hochberg FDR of less than .05 DE.

```
library(goseq)
genes = as.integer(p.adjust(tested$table$p.value, method = "BH") < 0.05)
names(genes) = row.names(tested$table)
```

Next, we calculate a probability weighting function, correcting for length bias, a technical bias present in all forms of RNA-seq data (see [3] for a details).

```
pwf=nullp(genes,'hg19','ensGene')
```

If we'd instead wanted to correct for total read count bias, we would calculate the pwf using the number of counts from each gene as follows:

```
pwf=nullp(genes,bias.data=rowsum(counts[match(names(genes),rownames(counts))]))
```

Finally, we calculate the p-value for each GO category being over represented amongst DE genes.

```
GO.pvals=goseq(genes,pwf,'hg19','ensGene')
```

Examples

This section provides an easy to follow example to illustrate the analysis pipeline outlined above.

Li Prostate cancer data set

Description

This data set compares prostate cancer LNcap cell lines with and without treatment by the testosterone like hormone androgen [4]. The sequencing was done using the Illumina GA I and produced 36 bp, single end, unstranded reads. The output from the machine are 7 files (each from a different sequencing lane):

untreated1.fa

untreated2.fa

untreated3.fa

untreated4.fa

treated1.fa

treated2.fa

treated3.fa

Note that this data set is slightly unusual in that the quality scores are missing from the reads. Therefore, we will have to keep this in mind when doing the analysis.

Quality Control

It's published data, so that's a pretty good quality control one would hope...

Sequence alignment

Building the reference

The first step in the pipeline is to align all the reads to a reference. As this data is taken from human LNcap cells, the latest build of the human genome is an obvious choice. We have installed BOWTIE (version 0.10.0) using all the standard options. There is a prebuilt copy of the human index available from the BOWTIE website, however, to illustrate building a genome from scratch we instead download the .fa files for the genome from the UCSC. We create a working directory containing the 7 RNA-seq data files and the file chromFA.tar.gz downloaded from http://hgdownload.cse.ucsc.edu/goldenPath/hg19/bigZips/.

The file chromFA.tar.gz contains the sequence of all human chromosomes, including the unallocated contigs. To make a BOWTIE index we need to concatenate them into a single file. We will exclude all the contigs from our fasta file.

```
tar -zxvf chromFA.tar.gz
```

We only want chr1-22.fa, chrX.fa, chrY.fa and chrM.fa, so delete everything else:

```
rm chr*_*.fa
```

Now we concatenate the desired files (it is useful to have the reference both in one chromosome per file and one file per genome formats, although the BOWTIE index can be made from either).

```
cat chr*.fa>hg19.fa
```

We build the BOWTIE index and name it hg19.

```
bowtie-build hg19.fa hg19
```

And move the BOWTIE index files to the appropriate location for BOWTIE to find them.

```
mv *.ebwt $BOWTIE_INDEXES
```

Aligning the reads

Having constructed the BOWTIE index for the human genome, we now proceed to map the reads from each lane. We want to use the --best and --sam flags. At this point we recall that our data lacks quality information for the reads. Therefore, we use the -v 3 option which ignores quality scores when aligning reads to the genome.

```
for src_fastafile in *treated*.fa
do
    bowtie -v 3 -p 8 --best --sam hg19 ${src_fastafile} ${src_fastafile%%.fa}.sam
done
```

The for loop is just for convenience and is equivalent to writing:

```
bowtie -v 3 -p 8 --best --sam hg19 untreated1.fa untreated1.sam
bowtie -v 3 -p 8 --best --sam hg19 untreated2.fa untreated2.sam
bowtie -v 3 -p 8 --best --sam hg19 untreated3.fa untreated3.sam
bowtie -v 3 -p 8 --best --sam hg19 untreated4.fa untreated4.sam
bowtie -v 3 -p 8 --best --sam hg19 treated1.fa treated1.sam
bowtie -v 3 -p 8 --best --sam hg19 treated2.fa treated2.sam
bowtie -v 3 -p 8 --best --sam hg19 treated3.fa treated3.sam
```

BOWTIE will output 7 SAM files containing the aligned reads. To count the fraction of aligned reads we run the following command at the shell (this information is also reported directly by BOWTIE, but it is useful to be able to calculate it yourself if need be).

```
awk '$3!="*"' untreated1.fa|wc -l
wc -l untreated1.fa
```

The first command prints the number of reads that have mapped in the BOWTIE output file, the second outputs 4 times the number of reads in the input file (because the fasta format is for 4 lines per read).

We will ignore those reads that cross exon-exon boundaries and continue with the analysis.

Summarization of reads

Converting to BAM

In order to summarize our aligned reads into genes, we first have to convert the SAM output of BOWTIE into the compressed, index BAM format. First we need to create an index for the human genome in the samtools format. For this we need the fasta file for the reference (which should be the same that was used to create the index for aligning the reads), which we already have, but we will reconstruct it from the BOWTIE index anyway.

```
bowtie-inspect hg19>hg19.fa
```

Now we construct the samtools index. The following command produces a .fai file which we can use to convert the SAM files to BAM files.

```
samtools faidx hg19.fa
```

Next, we convert all 7 SAM files to BAM files.

```
for SAM in *.sam
do
    #Convert to BAM format
    samtools import hg19.fa.fai ${SAM} ${SAM%%.sam}.bam
    #Sort everything
    samtools sort ${SAM%%.sam}.bam ${SAM%%.sam}_sorted
    #Create an index for fast searching
    samtools index ${SAM%%.sam}_sorted.bam
    #Delete temporary files
    rm ${SAM%%.sam}.bam
done
```

Again, the bash for loop is just for convenience, the above is equivalent to running the following for all 7 files:

```
samtools import hg19.fa.fai untreated1.sam untreated1.bam
samtools sort untreated1.bam untreated1_sorted
samtools index untreated1_sorted.bam
rm untreated1.bam
```

Processing in R

Next we need to load the sorted, indexed, BAM files into R. As we have unstranded RNA-seq, we need to make the strand designator for each read ambiguous (which is done by setting it to "*"). After starting R we run,

```
library(Rsamtools)
#Create a list of bam file object containing the short reads
bamlist=list()
src_files=list.files(pattern="*_sorted.bam$")
for(filename in src_files){
    #Since we do not know which strand the reads were originally transcribed,
    #so set the strand to be ambiguous
    tmp=readBamGappedAlignments(filename)
    bamlist[[length(bamlist)+1]]=GRanges(seqnames=rname(tmp),
        ranges=IRanges(start=start(tmp),end=end(tmp)),
        strand=rep("*",length(tmp)))
}
names(bamlist)=src_files
```

Having loaded the files into R, we next need to create an annotation object. Since we are using hg19, we can readily download one from the UCSC using the GenomicFeatures package. We choose to use the ENSEMBL gene annotation.

```
library(GenomicFeatures)
txdb=makeTranscriptDbFromUCSC(genome="hg19",tablename="ensGene")
```

We want to compare genes for differential expression, so we will summarize by gene and we choose to count all reads that fall within the body of the gene (including introns) as counting towards a genes count.

```
tx_by_gene=transcriptsBy(txdb,"gene")
```

Finally, we count the number of reads that fall in each gene for each lane and record the results in a table of counts.

```
#Initialize table of counts
toc=data.frame(rep(NA,length(tx_by_gene))
for(i in 1:length(bamlist)){
    toc[,i]=countOverlaps(tx_by_gene,bamlist[[i]])
}
#Fix up labels
rownames(toc)=names(tx_by_gene)
colnames(toc)=names(bamlist)
```

Differential Expression testing

Having finally obtained a table of counts, we now want to compare the treated and untreated groups and look for any statistically significant differences in the number of counts for each gene. We will do this using the negative binomial model used by edgeR.

Normalization

We calculate appropriate scaling factors for normalization using the TMM method with the first lane as the reference.

```
library(edgeR)
norm_factors=calcNormFactors(as.matrix(toc))
```

The counts themselves are not changed, instead these scale factors are used as an offset in the negative binomial model. This is incorporated in the DGE list object required by edgeR.

```
DGE=DGEList(toc,lib.size=norm_factors*colSums(toc),group=gsub("[0-9].*","",colnames(toc)))
```

Statistical Test

Next we calculate a common dispersion parameter which represents the additional extra Poisson variability in the data.

```
disp=estimateCommonDisp(DGE)
```

Which allows us to calculate p-values for genes being differentially expressed.

```
tested=exactTest(disp)
```

Gene Ontology testing

In order to test for over represented GO categories amongst DE genes, we first have to pick a cutoff for calling genes as differentially expressed after applying multiple hypothesis correction. We choose the ever popular cutoff for significance of .05

```
library(goseq)
#Apply benjamini hochberg correction for multiple testing
#choose genes with a p-value less than .05
genes=as.integer(p.adjust(tested$table$p.value,method="BH") <.05)
names(genes)=row.names(tested$table)
```

Now we calculate the probability weighting function, which quantifies the length bias effect.

```
pwf=nullp(genes,"hg19","ensGene")
```

Finally, we calculate the p-values for each GO category being over represented amongst DE genes.

```
GO.pvals=goseq(genes,pwf,"hg19","ensGene")
```

References

[1] Robinson MD and Oshlack A. A scaling normalization method for differential expression analysis of RNA-seq data. Genome Biology (2010), 11(3), R25

[2] Robinson MD*, McCarthy DJ* and Smyth GK (2010) edgeR: a Bioconductor package for differential expression analysis of digital gene expression data. Bioinformatics, 26(1):139-40

[3] Young MD, Wakefield MJ, Smyth GK and Oshlack A. Gene ontology analysis for RNA-seq: accounting for selection bias. Genome Biology (2010), 11(2), R14

[4] Li H, Lovci MT, Kwon YS, Rosenfeld MG, Fu XD, Yeo GW Determination of tag density required for digital transcriptome analysis: application to an androgen-sensitive prostate cancer model. Proc Natl Acad Sci USA 2008 , 105:20179-20184.

[5] Oshlack A, Robinson MD, Young MD. From RNA-seq reads to differential expression results. Genome Biology (2010), 11:220,