Exercises: A Simple RNA-seq Use Case

Patrick Aboyoun

Nicolas Delhomme

Ismael Padioleau

29-30 July, 2010

Contents

1 Introduction

This file describes an RNA-seq analysis use-case. RNA-seq[8] was introduced as a new method to perform Gene Expression Analysis, using the advantages of the high throughput of Next-Generation Sequencing machines.

The goal of this use-case is to generate a count table for the selected genetic features of interest, i.e. exons, transcripts, gene models, etc. In the first part, we will use Bioconductor packages *ShortRead*[7] and *GenomicFeatures* to define the counts for genetic features. This information will then be exported into a wig formatted file for visualization in the UCSC genome browser or a stand-alone genome browser like IGB.

2 Single sample use case

In this section, we will demonstrate how to generate a count table containing the number of sequencing reads that can be assigned to given genetic features. An expressed genetic feature can be anything from an exon to a gene-model and, as recently published, enhancers[4]. In this section, we will generate such a count table for a single sample.

2.1 Reading the data

We read the export file, using the *ShortRead* package and have a look at its contents.

```
> library(ShortRead)
```

```
> aln <-
+   readAligned(system.file("extdata", "subset_export.txt.gz",
+                           package = "HTSandGeneCentricLabs"),
+               type = "SolexaExport")
> aln
```

```
class: AlignedRead
length: 100000 reads; width: 36 cycles
chromosome: NM 1:0:0 ... chr2R chr2L
position: NA NA ... 20555556 13903608
strand: NA NA ... + -
alignQuality: NumericQuality
alignData varLabels: run lane ... filtering contig
```

2.2 Filtering the data

Illumina uses a built-in chastity filter that describes whether a read cluster could be successfully sequenced, with `Y` representing yes and `N` representing no.

```
> table(chastity = alignData(aln)[["filtering"]],
+       aligned = is.na(position(aln)),
+       useNA = "ifany")
```

```
        aligned
chastity FALSE  TRUE
       Y 56883  7323
       N 12065 23729
```

In this sample, roughly 36% of the reads do not pass the chastity filter and an additional 7.3% do not align to the genome. Also, some of these reads contain many "N"s, which occurs whenever Bustard, the Illumina base caller, could not produce a valid base call. All these reads are questionable and can be filtered out.

We will use three types of filters on our data:

1. Keep reads with at most 2 Ns.
2. Keep only the alignments to chromosomes chr2L, chr2R, chr3L, chr3R, and chr4.
3. Use Illumina's chastity filter.

```
> nFilt <- nFilter(2)
> chrFilt <- chromosomeFilter(regex = "chr[0-9]")
> library(HTSandGeneCentricLabs)
> cFilt <- alignDataFilter(expression(filtering=="Y"))
> filt <- compose(nFilt, chrFilt, cFilt)
> aln <- aln[filt(aln)]
> aln
```

```
class: AlignedRead
length: 46070 reads; width: 36 cycles
chromosome: chr3L chr3L ... chr2R chr2L
position: 9861757 21533621 ... 20555556 13903608
strand: + + ... + -
alignQuality: NumericQuality
alignData varLabels: run lane ... filtering contig
```

We are now left with 46,070 "valid" alignments, which we want to assign to their respective exon. For this we need to get the proper genomic and genetic information.

The *AlignedRead* contain is not idea for performing interval overlap calculations, so we will create a *GRanges* instance that contains the alignment locations. Since this RNA-seq protocol could not discern strand information, we will set all strand locations to the wildcard *.

```
> alnRanges <- as(aln, "GRanges")
> strand(alnRanges) <- "*"
```

2.3 Obtaining the transcript annotation

To assign the alignments to their respective exons, we need to know the genome composition of the model organism "Drosophila melanogaster". We can obtain this information using the GenomicFeatures, which can create annotation databases using either Biomart or UCSC as a resource. For the purposes of our analysis, we will use Biomart.

```
> library(GenomicFeatures)
```

The code to make this from Biomart is

```
> dmTxDb <-
+   makeTranscriptDbFromBiomart(biomart = "ensembl",
+                               dataset =
+                               "dmelanogaster_gene_ensembl")
```

but we will load a local copy for expediency.

```
> dmTxDb <-
+   loadFeatures(system.file("extdata", "dmTxDb.sqlite",
+                            package = "HTSandGeneCentricLabs"))
> dmTxDb

TranscriptDb object:
| Db type: TranscriptDb
| Data source: BioMart
| BioMart database: ensembl
| BioMart dataset: dmelanogaster_gene_ensembl
| BioMart dataset description: Drosophila melanogaster genes (BDGP5.13)
| BioMart dataset version: BDGP5.13
| Full dataset: yes
| transcript_nrow: 22423
| exon_nrow: 70767
| cds_nrow: 61189
| Db created by: GenomicFeatures package from Bioconductor
| Creation time: 2010-06-09 21:43:51 -0700 (Wed, 09 Jun 2010)
| GenomicFeatures version at creation time: 1.0.0
| RSQLite version at creation time: 0.9-1
```

As is typical, the coding for the sequence names differ between the experiment data and the annotation metadata, so we will recode the seqnames in the alignments to coincide with those in the transcript database.

```
> levels(seqnames(alnRanges))

[1] "chr2L" "chr2R" "chr3L" "chr3R" "chr4"

> seqnames(dmTxDb)

 [1] "dmel_mitochondrion_genome" "2RHet"
 [3] "2L"                        "3L"
 [5] "X"                         "XHet"
 [7] "Uextra"                    "4"
 [9] "YHet"                      "U"
[11] "2LHet"                     "3R"
[13] "3RHet"                     "3LHet"
[15] "2R"

> seqnames(alnRanges) <- sub("^chr", "", seqnames(alnRanges))
> seqlengths(alnRanges) <-
+   seqlengths(dmTxDb)[names(seqlengths(alnRanges))]
> head(alnRanges, 3)
```

```
GRanges with 3 ranges and 7 elementMetadata values
    seqnames               ranges strand |      run
       <Rle>            <IRanges>  <Rle> | <factor>
[1]       3L [ 9861757,  9861792]      * |    90320
[2]       3L [21533621, 21533656]      * |    90320
[3]       3R [25871248, 25871283]      * |    90320
         lane      tile         x         y filtering
    <integer> <integer> <integer> <integer>  <factor>
[1]         3         1         0       990         Y
[2]         3         1         0       452         Y
[3]         3         1         0       965         Y
      contig
    <factor>
[1]
[2]
[3]

seqlengths
       2L       2R       3L       3R        4
 23011544 21146708 24543557 27905053  1351857
```

2.4 Analysis of gene models

Now that we have the alignment locations and the transcript annotations, we can begin with a coarse model of a gene. For simplicity we will start with the boundaries for each gene based upon the transcripts in the `dmTxDb` object. The *HTSandGeneCentricLabs* package contains the function `geneBounds` for this purpose. Since we have limited our examination to chromosomes 2L, 2R, 3L, 3R, and 4 in fly.

```
> dmGeneBounds <- HTSandGeneCentricLabs::geneBounds(dmTxDb)
> dmGeneBounds <-
+   dmGeneBounds[seqnames(dmGeneBounds) %in%
+                levels(seqnames(alnRanges))]
> head(dmGeneBounds, 3)

GRanges with 3 ranges and 0 elementMetadata values
            seqnames               ranges strand |
               <Rle>            <IRanges>  <Rle> |
FBgn0000003       3R [ 2648220,  2648518]      + |
FBgn0000008       2R [18024494, 18060346]      + |
FBgn0000014       3R [12633349, 12655769]      - |

seqlengths
 dmel_mitochondrion_genome ...                       2R
                     19517 ...                 21146708
```

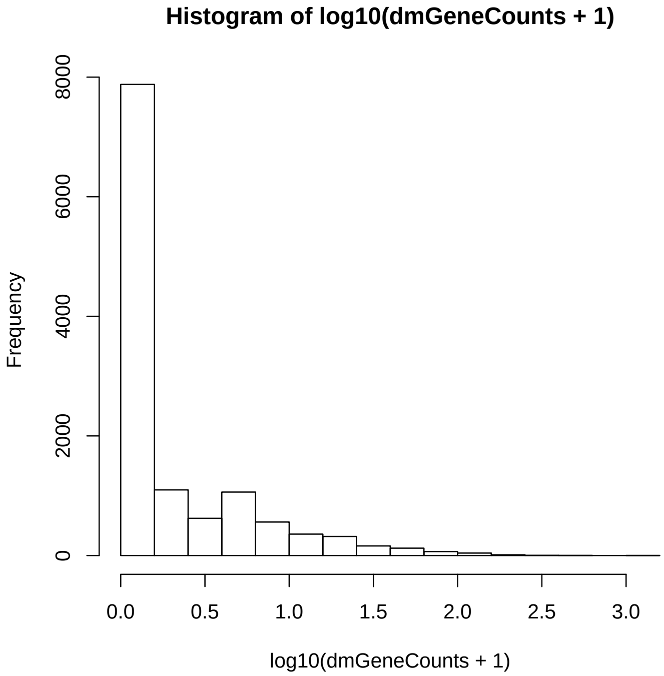

Figure 1: Distribution of gene model interval overlaps count on the log scale.

In order to determine expression levels for each of these genes we will count the number of interval overlaps that occur with the aligned ranges. This can be achieved using the `countOverlaps` method from the *GenomicRanges* package.

```
> dmGeneCounts <- countOverlaps(dmGeneBounds, alnRanges)
> names(dmGeneCounts) <- names(dmGeneBounds)
> hist(log10(dmGeneCounts+1))
```

2.5 Normalizing counts

A common way to normalize reads is to convert them to RPKM: reads per million reads in the library per kb of transcripts. This implies normalizing the read counts depending on the genic feature size (exon, transcript, gene model,...) and depending on the total number of reads sequenced for that library. *HTSandGeneCentricLabs* count tables can be easily transformed into RPKM, by using the `rpkm` method:

```
> dmRPKM <- HTSandGeneCentricLabs::rpkm(dmGeneCounts, dmGeneBounds)
> hist(log10(dmRPKM+1))
```

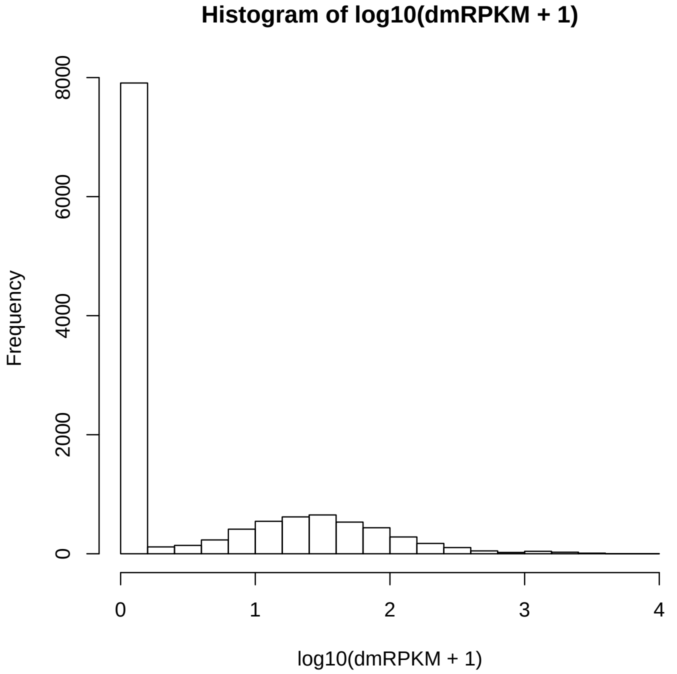

Figure 2: Distribution of RPKM on the log scale.

But such a count normalization is sub-optimal, as you would “limit” your dynamic range to the one of the library having the lowest number of reads. A better way of normalizing the data is to use either the *edgeR*[10] or *DESeq*[2] packages. The approaches taken by this packages are explained in their respective vignettes.

2.6 Differentiating amongst isoforms

Once we have used other packages such as *edgeR* and *DESeq*, we can use basic Bioconductor tools to find the interval overlap counts for each of the exons within the known isoforms of the annotated genes. This counting activity can be achieved using the `countExonIdsByTxOverlaps` function within the *HTSandGeneCentricLabs* package.

```
> dmExonIdsByTxCounts <-
+   HTSandGeneCentricLabs::countExonIdsByTxOverlaps(dmTxDb,
+                                                   alnRanges)
> dmExonIdsByTxCounts
```

```
CompressedSplitDataFrameList of length 0
<0 elements>
```

The output of `countExonIdsByTxOverlaps` is more complex than the typical R object. It returns a *CompressedSplitDataFrameList* that is split by gene ID, where each row in the split represent a transcript. These data rows are comprised of three columns: the transcipt id, the number of interval overlaps for each exon within the spliced transcript, and the total number of interval overlaps across the entire spliced transcript. The elements in this *CompressedSplitDataFrameList* object are sorted in descending order by the transcript with the largest total interval overlaps.

2.7 *De novo* transcript identification

In theory, RNA-seq experiments can be used to identify any transcribed molecule, since the technique is not dependent on a predefined sets of probes like microarrays are. Therefore, RNA-seq is a potential useful tool in finding unknown transcripts and isoforms, as well as regulatory transcribed elements. To that end, several methods are available to recreate and annotate transcripts, e.g. Oases, Velvet[13, 14], TopHat[12], to cite some of them (see [1] as well), but few have been done for other regulatory transcribed elements such as eRNAs [4]. We can use Bioconductor tools to identify locus and quantify counts without prior annotation knowledge.

The process begins with calculating the coverage, using the method from the *GenomicRanges* package.

```
> cover <- coverage(alnRanges)
> head(cover, 2)
```

```
SimpleRleList of length 2
$`2L`
'integer' Rle of length 23011544 with 17850 runs
  Lengths:  6777    36  2316    36 ...   499    36 50474
  Values :     0     1     0     1 ...     0     1     0

$`2R`
'integer' Rle of length 21146708 with 21751 runs
  Lengths: 19042    36    36    36 ...  2574    36  2520
  Values :     0     1     0     1 ...     0     1     0
```

Next the islands can be formed using the `slice` function. The peak height for the islands can be found using the `viewMaxs` function and the island widths can be found using the `width` function.

```
> islands <- slice(cover, 1)
> islandPeakHeight <- viewMaxs(islands)
> islandWidth <- width(islands)
> median(islandPeakHeight)

2L 2R 3L 3R  4
 1  1  1  1  1

> median(islandWidth)

2L 2R 3L 3R  4
36 36 36 36 36
```

While some sophisticated bioinformatic approaches can be taken to find exons *de novo* from the RNA-seq sample, we can use a simple approach whereby we select islands whose maximum peak height is 2 or more and whose width is 54 bp or more. The `elementLengths` function calls shows how many of these candidate exons appear on each chromosome.

```
> candidateExons <-
+   islands[islandPeakHeight >= 1L & islandWidth >= 54L]
> elementLengths(candidateExons)

  2L   2R   3L   3R    4
 740  872  732 1095   38
```

2.8 Exporting the coverage

If we wish to visualize the coverage of the reads on the genome in a genome browser, we can use the *rtracklayer* package to export the coverage to a wig file.

```
> library(rtracklayer)
> names(cover) <- paste("chr", names(cover), sep="")
> export(cover["chr4"], con = "chr4.wig")
```

You can now visualize it in a genome browser of your choice. This is a common way to assess if the raw data you are looking at agrees with the experimental design that was performed. However, this is still far from being able to normalize and call differential expression between different conditions, which is usually the goal of RNA-seq experiments. For more information on the analysis issues, see the *DESeq*[2] Bioconductor package.

3 De-multiplexing sample use case

Nowadays, NGS machines produces so many "raw" reads (40M for Illumina, 100M for SOLiD), that the coverage obtained per lane for the transcriptome of "small" genome-sized organisms, is way too big. Therefore, techniques to have several samples running as part of the same library have been created[5, 11], using 6bp barcodes to uniquely identify the sample. This is called multiplexing and one can today with an average Illumina GenomeAnalyzer GAIIx average run, multiplex 12 yeast samples and even 2 drosophila samples in a single lane. Actually, if the lane is very good (30M aligning reads), one can multiplex 4 of them. This approach is very advantageous for researchers, especially in term of costs, but it adds an additional layer of pre-processing that is not as trivial to process as one would have thought. Extracting the barcodes is fairly straightforward, however the average 1 percent sequencing error rate introduces a lot of multiplicity in the actual barcodes present in the samples and this needs to be sorted out accordingly. A proper design of the barcodes, maximizing the Hamming distance (http://en.wikipedia.org/wiki/Hamming_distance) is an essential step for a proper de-multiplexing.

There are two kinds of barcoding, the one described in Lefrancois *et al.* [5] where the barcode is part of the read sequence and the one developed by Illumina, where the barcode is read in a separate sequencing reaction after the first mate sequencing.

```
> alns <-
+   readAligned(system.file("extdata", "multiplex_export.txt.gz",
+                           package = "HTSandGeneCentricLabs"),
+               filter =
+               compose(alignDataFilter(expression(filtering=="Y")),
+                       nFilter(2),
+                       chromosomeFilter(regex = "chr[0-9]")),
+               type = "SolexaExport",
+               withAll = TRUE)
> indexes <- c("ACACTG", "ACTAGC", "ATGGCT", "TTGCGA")
> demultAlns <-
+   demultiplex(alns, indexes=indexes, edit.dist=2, indexes.qty=4,
+               type="within")
```

4 You are done, but still there is more to come...

It is known that the standard Illumina RNA-seq protocol shows a bias in the first 12 nucleotides of every read. It is still unclear where this bias comes from (fragmentation, random hexamer priming, RNase H sequence specificity), but there has been a couple of publication recently that proposes corrections for that bias [6, 3]. We anticipate creating some Bioconductor functionality to correct this bias in the near future.

5 Session Information

The version number of R[9] and packages loaded for generating the vignette were:

```
> toLatex(sessionInfo())
```

- R version 2.11.1 Patched (2010-07-25 r52612), `i386-apple-darwin9.8.0`
- Locale: `C/C/C/C/C/en_US.UTF-8`
- Base packages: base, datasets, grDevices, graphics, methods, stats, tools, utils
- Other packages: AnnotationDbi 1.10.2, Biobase 2.8.0, Biostrings 2.16.9, DBI 0.2-5, EatonEtAlChIPseq 0.0.1, GO.db 2.4.1, GenomicFeatures 1.0.6, GenomicRanges 1.0.7, HTSandGeneCentricLabs 0.0.3, IRanges 1.6.11, KEGG.db 2.4.1, RCurl 1.4-3, RSQLite 0.9-2, Rsamtools 1.0.7, ShortRead 1.6.2, bitops 1.0-4.1, hgu95av2.db 2.4.1, lattice 0.18-8, org.Hs.eg.db 2.4.1, rtracklayer 1.8.1
- Loaded via a namespace (and not attached): BSgenome 1.16.5, XML 3.1-0, biomaRt 2.4.0, grid 2.11.1, hwriter 1.2

References

[1] Adam Ameur, Anna Wetterbom, Lars Feuk, and Ulf Gyllensten. Global and unbiased detection of splice junctions from rna-seq data. *Genome Biology 2010 11:202*, 11(3):R34, Mar 2010.

[2] Simon Anders and Wolfgang Huber. Differential expression analysis for sequence count data.

[3] Kasper D Hansen, Steven E Brenner, and Sandrine Dudoit. Biases in illumina transcriptome sequencing caused by random hexamer priming. *Nucleic Acids Research*, Apr 2010.

[4] Tae-Kyung Kim, Martin Hemberg, Jesse M Gray, Allen M Costa, Daniel M Bear, Jing Wu, David A Harmin, Mike Laptewicz, Kellie Barbara-Haley, Scott Kuersten, Eirene Markenscoff-Papadimitriou, Dietmar Kuhl, Haruhiko Bito, Paul F Worley, Gabriel Kreiman, and Michael E Greenberg. Widespread transcription at neuronal activity-regulated enhancers. *Nature*, 465(7295):182–7, May 2010.

[5] Philippe Lefrançois, Ghia M Euskirchen, Raymond K Auerbach, Joel Rozowsky, Theodore Gibson, Christopher M Yellman, Mark Gerstein, and Michael Snyder. Efficient yeast chip-seq using multiplex short-read dna sequencing. *BMC Genomics*, 10:37, Jan 2009.

[6] Jun Li, Hui Jiang, and Wing Hung Wong. Modeling non-uniformity in short-read rates in rna-seq data. *Genome Biology 2010 11:202*, 11(5):R50, May 2010.

[7] Martin Morgan, Simon Anders, Michael Lawrence, Patrick Aboyoun, Hervé Pagès, and Robert Gentleman. Shortread: a bioconductor package for input, quality assessment and exploration of high-throughput sequence data. *Bioinformatics*, 25(19):2607–8, Oct 2009.

[8] Ali Mortazavi, Brian A Williams, Kenneth Mccue, Lorian Schaeffer, and Barbara Wold. Mapping and quantifying mammalian transcriptomes by rna-seq. *Nature Methods*, 5(7):621–8, Jul 2008.

[9] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2009. ISBN 3-900051-07-0.

[10] Mark D Robinson, Davis J McCarthy, and Gordon K Smyth. edger: a bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics*, 26(1):139–40, Jan 2010.

[11] Andrew M Smith, Lawrence E Heisler, Robert P St Onge, Eveline Farias-Hesson, Iain M Wallace, John Bodeau, Adam N Harris, Kathleen M Perry, Guri Giaever, Nader Pourmand, and Corey Nislow. Highly-multiplexed barcode sequencing: an efficient method for parallel analysis of pooled samples. *Nucleic Acids Research*, May 2010.

[12] C Trapnell, L Pachter, and S. L Salzberg. Tophat: discovering splice junctions with rna-seq. *Bioinformatics*, 25(9):1105–1111, May 2009.

[13] Daniel R Zerbino and Ewan Birney. Velvet: algorithms for de novo short read assembly using de bruijn graphs. *Genome Research*, 18(5):821–9, May 2008.

[14] Daniel R Zerbino, Gayle K McEwen, Elliott H Margulies, and Ewan Birney. Pebble and rock band: heuristic resolution of repeats and scaffolding in the velvet short-read de novo assembler. *PLoS ONE*, 4(12):e8407, Jan 2009.