Working with the ShortRead Package

James Bullard Kasper D. Hansen

Modified: April 18, 2010. Compiled: October 30, 2017

In this document we show how to use the *Genominator* package with the *ShortRead* package and conduct a simple differential expression analysis.

1 Importing data

```
> require(Genominator)
> require(ShortRead)
> require(yeastRNASeq)
```

The data we will use are described in the vignette of the *yeastRNASeq* and was originally published in [1]. But to summarize, we have data from two different yeasts, a wild-type strain ("`wt`") and a mutant strain ("`mut`"). Part of the RNA degradation pathway is knocked out in the mutant. For each strain we have two lanes worth of data – it is the exact same sample and library preparation sequenced in both lanes. Only 500,000 raw reads from each lane is part of the *yeastRNASeq* package and they have been aligned with Bowtie, keeping only unique hits with up to two mismatches. Since not all reads align, we will be working with a up to 500,000 reads per lane.

The data is available as a list of *AlignedRead* class objects:

```
> data(yeastAligned)
> yeastAligned[[1]]
```

```
class: AlignedRead
length: 423318 reads; width: 26 cycles
chromosome: Scchr05 Scchr15 ... Scchr08 Scchr13
position: 541317 885627 ... 488228 667296
strand: - + ... - +
alignQuality: NumericQuality
alignData varLabels: similar mismatch
```

```
> sapply(yeastAligned, length)
```

```
mut_1_f mut_2_f  wt_1_f  wt_2_f
 423318  420848  410349  430264
```

In the package we also have the Bowtie output files, which we will use for illustration:

```
> list.files(file.path(system.file(package = "yeastRNASeq"), "reads"),
+            pattern = "bowtie")
```

```
[1] "mut_1_f.bowtie.gz" "mut_2_f.bowtie.gz" "wt_1_f.bowtie.gz"
[4] "wt_2_f.bowtie.gz"
```

The Bowtie output files are compressed using gzip, *ShortRead* can handle this.

Before importing a set of aligned reads using *Genominator* one usually has to make some decisions regarding chromosome names as well as names of the columns in the resulting database.

Note the chromosome names are a bit special: `Scchr01`. In our experience it is quite common for different annotation sources to use different shorthands for the chromosomes (for yeast we have at least seen `chr1`, `chr01`, `Scchr01`, `chrI`). This is a bit of a pain and is very hard to automate. In *Genominator* we expect the user to explicitly convert the chromosome names to a common representation and we furthermore require chromosome names to be integers. For the sake of importing alignment files this is accomplished by using the `chrMap` argument that is a simple character vector which the chromosome names are matched up against. The first element in the vector gets assigned the integer 1 and so on. If there are chromosomes in the alignment file that does not appear in the `chrMap` vector, the corresponding reads are not imported. This is often useful, but can also lead to loss of data.

We construct the `chrMap` object by

```
> chrMap <- paste("Scchr", sprintf("%02d", 1:16), sep = "")
> unique(chromosome((yeastAligned[[1]])))
```

```
 [1] Scchr05 Scchr15 Scchr13 Scchr16 Scchr01 Scchr14 Scchr12 Scchr03 Scchr07
[10] Scchr11 Scmito  Scchr10 Scchr08 Scchr04 Scchr09 Scchr06 Scchr02
17 Levels: Scchr01 Scchr02 Scchr03 Scchr04 Scchr05 Scchr06 Scchr07 ... Scmito
```

We see that by using this version of `chrMap` we will drop reads aligning to the mitochondrial chromosome.

The other decision we need to make is how to map filenames to names of the columns of the resulting database. Note that whatever choice we make is more or less permanent. Usually we prefer short, descriptive names. It makes a lot of sense to construct these names programmatic from the filenames:

```
> files <- list.files(file.path(system.file(package = "yeastRNASeq"), "reads"),
+                     pattern = "bowtie", full.names = TRUE)
> names(files) <- sub("_f\\.bowtie\\.gz", "", basename(files))
> names(files)
```

```
[1] "mut_1" "mut_2" "wt_1"  "wt_2"
```

The way this is all specific is through a named vector of filenames, the names of the vector corresponds to column names in the resulting database. If two (or more) file entries have the same name, they will be joined into one column.

We now import the alignment files using `importFromAlignedReads`, which uses the `readAligned` function from *ShortRead* to parse the files.

```
> eData <- importFromAlignedReads(files, chrMap = chrMap,
+                                 dbFilename = "my.db",
+                                 tablename = "raw", type = "Bowtie")
> eData
```

```
table: raw

database file:
C:\Users\biocbuild\bbs-3.6-bioc\tmpdir\RtmpE3N232\Rbuild1b8c291341a7\Genominator\vignettes\my.db


index columns: chr location strand

mode: w

schema:

      chr  location    strand     mut_1     mut_2      wt_1      wt_2
"INTEGER" "INTEGER" "INTEGER" "INTEGER" "INTEGER" "INTEGER" "INTEGER"
```

```
> head(eData)
```

```
  chr location strand mut_1 mut_2 wt_1 wt_2
1   1     3888      1     1    NA   NA   NA
2   1     3970      1    NA    NA    1   NA
3   1     3988      1    NA     1   NA   NA
4   1     4126     -1    NA    NA   NA    1
5   1     4242      1     1    NA   NA   NA
6   1     4296     -1     1    NA   NA   NA
```

This will create a database `my.db` in the current directory. The reads are automatically associated to the genomic location corresponding to the 5' end of the read – this is slightly different from some other programs that uses the lefternmost location of the read (the difference is in reads mapping to the reverse strand).

2 Annotation

There are at least two easy ways to retrieve annotation using Bioconductor: the *biomaRt* package which accesses Ensembl and the *rtracklayer* package which accesses the UCSC genome browser. There are oftentimes species-specific databases (for yeast we have at least SGD) which is a third source of annotation.

Using biomaRt

We will use *biomaRt* to retrieve information from the Ensembl database. We will illustrate a few pitfalls by comparing two different, but similar looking queries. More information on using *biomaRt* can be found it its excellent vignette.

The following code was run in January 2010.

```
> require(biomaRt)
> mart <- useMart("ensembl", "scerevisiae_gene_ensembl")
> attributes.gene <- c("ensembl_gene_id", "chromosome_name",  "start_position",
+                      "end_position", "strand", "gene_biotype")
> attributes.tr <- c("ensembl_gene_id", "ensembl_transcript_id", "ensembl_exon_id", "chromosome_name",
+                    "end_position", "strand", "gene_biotype", "exon_chrom_start", "exon_chrom_end", "ra
> ensembl.gene <- getBM(attributes = attributes.gene,  mart = mart)
> ensembl.transcript <- getBM(attributes = attributes.tr,  mart = mart)
```

The output is saved in the `yeastAnno.sources` object which is a list containing various annotation objects from yeast.

```
> data(yeastAnno.sources)
> ensembl.gene <- yeastAnno.sources$ensembl.gene
> ensembl.transcript <- yeastAnno.sources$ensembl.transcript
> head(ensembl.gene, n = 2)

  ensembl_gene_id chromosome_name start_position end_position strand
1         YHR055C            VIII         214535       214720     -1
2         YPR161C             XVI         864445       866418     -1
    gene_biotype
1 protein_coding
2 protein_coding

> head(ensembl.transcript, n = 2)

  ensembl_gene_id ensembl_transcript_id ensembl_exon_id chromosome_name
1         YHR055C               YHR055C       YHR055C.1            VIII
2         YPR161C               YPR161C       YPR161C.1             XVI
  start_position end_position strand   gene_biotype exon_chrom_start
```

```
1         214535       214720     -1 protein_coding           214535
2         864445       866418     -1 protein_coding           864445
  exon_chrom_end rank
1         214720    1
2         866418    1
```

```
> dim(ensembl.gene)
```

```
[1] 7124    6
```

```
> dim(ensembl.transcript)
```

```
[1] 7547   11
```

```
> subset(ensembl.gene, ensembl_gene_id == "YPR098C")
```

```
  ensembl_gene_id chromosome_name start_position end_position strand
7         YPR098C             XVI         728945       729526     -1
    gene_biotype
7 protein_coding
```

```
> subset(ensembl.transcript, ensembl_gene_id == "YPR098C")
```

```
  ensembl_gene_id ensembl_transcript_id ensembl_exon_id chromosome_name
7         YPR098C               YPR098C       YPR098C.1             XVI
8         YPR098C               YPR098C       YPR098C.2             XVI
  start_position end_position strand   gene_biotype exon_chrom_start
7         728945       729526     -1 protein_coding           729480
8         728945       729526     -1 protein_coding           728945
  exon_chrom_end rank
7         729526    1
8         729383    2
```

```
> length(unique(ensembl.transcript$ensembl_transcript_id))
```

```
[1] 7124
```

Note that altering the query a bit leads to a quite different number of rows being retrieved. In this case it is because each row in one query correspond to a gene, whereas each row in the other query correspon to an exon of a transcript.

A key observation here is that the columns `start_position` and `end_position` contains the start and end position of the gene which is different from the start and end position of the exon when we look at a gene consiting of two exons. Furthermore, it is not even clear from the output of the first query that genes with multiple exons exists. In this example, there is no difference between the transcript id and the gene id, because (at least according to this annotation), there are no genes in yeast that produces multiple transcripts.

In this case, we would argue that the right object to use is `yeastAnno.transcripts`. Below we will post-process this object for use with *Genominator*.

Using rtracklayer

Here we will use the *rtracklayer* package to access the UCSC genome browser. UCSC sometimes have different tables for a specific genome. We will take a closer look at the SGD table (based on the name we presume that it is supposed to package information from SGD (= Saccharomyces Genome Database)) and the ENS table (which we assume is an attempt to package information from Ensembl)

The following code was run in January 2010

```
> require(rtracklayer)
> session <- browserSession()
> genome(session) <- "sacCer2"
> ucsc.sgdGene <- getTable(ucscTableQuery(session, "sgdGene"))
> ucsc.ensGene <- getTable(ucscTableQuery(session, "ensGene"))
```

We will also examine the output from these two tables

```
> data(yeastAnno.sources)
> ucsc.sgdGene <- yeastAnno.sources$ucsc.sgdGene
> ucsc.ensGene <- yeastAnno.sources$ucsc.ensGene
> head(ucsc.sgdGene, n = 2)

  bin    name chrom strand txStart  txEnd cdsStart cdsEnd exonCount exonStarts
1  73 YAL012W  chrI      +  130801 131986   130801 131986         1    130801,
2 585 YAL069W  chrI      +     334    649      334    649         1       334,
  exonEnds proteinID
1  131986,    P31373
2     649,       n/a

> head(ucsc.ensGene, n = 2)

  bin    name chrom strand txStart  txEnd cdsStart cdsEnd exonCount exonStarts
1  73 YAL012W  chrI      +  130801 131986   130801 131986         1    130801,
2 585 YAL069W  chrI      +     334    649      334    649         1       334,
  exonEnds score   name2 cdsStartStat cdsEndStat exonFrames
1  131986,     0 YAL012W         cmpl       cmpl         0,
2     649,     0 YAL069W         cmpl       cmpl         0,

> dim(ucsc.sgdGene)

[1] 6717   12

> dim(ucsc.ensGene)

[1] 7124   16

> subset(ucsc.sgdGene, name == "YPR098C")

     bin    name  chrom strand txStart  txEnd cdsStart cdsEnd exonCount
5756 590 YPR098C chrXVI      -  728944 729526   728944 729526         2
         exonStarts       exonEnds proteinID
5756 728944,729479, 729383,729526,    Q06089

> subset(ucsc.ensGene, name == "YPR098C")

     bin    name  chrom strand txStart  txEnd cdsStart cdsEnd exonCount
6104 590 YPR098C chrXVI      -  728944 729526   728944 729526         2
         exonStarts       exonEnds score   name2 cdsStartStat cdsEndStat
6104 728944,729479, 729383,729526,     0 YPR098C         cmpl       cmpl
     exonFrames
6104       2,0,

> subset(ucsc.sgdGene, name == "YER102W")

    bin    name chrom strand txStart  txEnd cdsStart cdsEnd exonCount
373 587 YER102W  chrV      +  362728 363698   363095 363698         2
        exonStarts       exonEnds proteinID
373 362728,363095, 363088,363698,    P05754
```

```
> subset(ucsc.ensGene, name == "YER102W")
```

```
    bin    name chrom strand txStart  txEnd cdsStart cdsEnd exonCount
424 587 YER102W  chrV      +  363095 363698   363095 363698         1
    exonStarts exonEnds score   name2 cdsStartStat cdsEndStat exonFrames
424    363095,  363698,     0 YER102W         cmpl       cmpl         0,
```

From this we see that the two tables have quite a different number of genes in them, that the two tables look very similar on the two exon gene considered in the previous section (although one table has the additional information of a protein ID), but that the two tables differ on at least one gene (actually, there are 757 entries that have same value in the `name` column but different values in the `exonStarts` column).

Some comments on annotation

As we see here, there are different sources of annotation that differ, even for a relatively simple and well-studied species as *S. cerevisiae*. We cannot give any recommendation as to what annotation source to use, that depends on the biological question and possibly other factors.

However, some effort ought to be spend on making sure that the used annotation matches up with the genome used. Even for yeast there are several different genomes. And while they differ in only a few substitutions and insertion/deletions, the insertion/deletions can easily lead to "off-by-a-little" errors.

Finally we note that for this specific example, none of the queries above display the SGD classification of genes into "verified", "dubious", and "uncharacterized", a classification that is often important when analyzing ones results. This information is obtainable directly from SGD and perhaps from a better use of the annotation tools above.

Post-processing the annotation

In the following we will work with the `ensembl.transcript` object, which we now post-process for use with *Genominator*.

In *Genominator* an annotation object is a *data.frame* with columns `chr, start, end, strand` where `chr` is an integer (and should match up with whatever was used in the import step earlier) and `strand` has values in $\{-1, 1, 0\}$ with 0 indicating that there is no strand information (and hence 0 matches both 1 and -1).

```
> yAnno <- yeastAnno.sources$ensembl.transcript
> yAnno$chr <- match(yAnno$chr, c(as.character(as.roman(1:16)), "MT", "2-micron"))
> yAnno$start <- yAnno$start_position
> yAnno$end <- yAnno$end_position
> rownames(yAnno) <- yAnno$ensembl_exon_id
> yAnno.simple <- yAnno[yAnno$chr %in% 1:16, c("chr", "start", "end", "strand")]
> head(yAnno.simple, n = 2)
```

```
          chr  start    end strand
YHR055C.1   8 214535 214720     -1
YPR161C.1  16 864445 866418     -1
```

```
> head(yAnno, n = 2)
```

```
          ensembl_gene_id ensembl_transcript_id ensembl_exon_id chromosome_name
YHR055C.1         YHR055C               YHR055C       YHR055C.1            VIII
YPR161C.1         YPR161C               YPR161C       YPR161C.1             XVI
          start_position end_position strand   gene_biotype exon_chrom_start
YHR055C.1         214535       214720     -1 protein_coding           214535
YPR161C.1         864445       866418     -1 protein_coding           864445
          exon_chrom_end rank chr  start    end
YHR055C.1         214720    1   8 214535 214720
YPR161C.1         866418    1  16 864445 866418
```

(note that we remove all annotation on the mitochondria and the plasmid).

A useful function is `validAnnotation` that checks whether the produced `data.frame` satisfy the annotation assumptions

```
> validAnnotation(yAnno)
```

```
[1] TRUE
```

3 Gene level counts

It is easy to obtain region level counts for a given annotation object:

```
> geneCounts.1 <- summarizeByAnnotation(eData, yAnno, ignoreStrand = TRUE)
> head(geneCounts.1)
```

```
          mut_1 mut_2 wt_1 wt_2
YHR055C.1     0     0    0    0
YPR161C.1    38    39   35   34
YOL138C.1    31    34   40   27
YDR395W.1    54    54   47   46
YGR129W.1    28    26    4    5
YPR165W.1   184   177  150  181
```

This produces a matrix containing the read counts per gene. Such a matrix is ready for analysis by various packages as well as the functionality in *Genominator*.

We use `ignoreStrand = TRUE` because the experimental assay does not keep strand of origin, so we count reads on either strand, inside each atomic region.

Note that the resulting matrix has one row for each row in the annotation object. Inthis annotation object, rows correspond to exons. We can get gene level counts either by using a `tapply` or directly from *Genominator* by

```
> geneCounts.2 <- summarizeByAnnotation(eData, yAnno, ignoreStrand = TRUE,
+                                       groupBy = "ensembl_gene_id")
> head(geneCounts.2)
```

```
         mut_1 mut_2 wt_1 wt_2
15S_rRNA     0     0    0    0
21S_rRNA     0     0    0    0
HRA1         7    14    4   12
LSR1       389   402   50   60
NME1       182   170    7    6
Q0010        0     0    0    0
```

Note that – because of the way reads are represented in the database when we use `importFromAligned-Reads` – that a read is counting as part of a gene if the first sequenced base maps within the region defined by the gene. This may create some concern at gene boundaries.

There are further complications. In yeast, it is very common for two genes to overlap each other on opposite strands. In other organisms, a gene may have several transcripts. For this purpose, *Genominator* supports the computation of various gene models.

Union-intersection (UI) genes were introduced by []. The UI representation of a gene is the set of bases that are annotated as being part of every transcript of the gene and that are not part of any transcript of any other gene.

We may transform our annotation object into a UI representation by

```
> yAnno.UI <- makeGeneRepresentation(yAnno, type = "UIgene", gene.id = "ensembl_gene_id",
+                                    transcript.id = "ensembl_transcript_id")
> head(yAnno.UI)
```

```
  chr  start    end strand ensembl_gene_id
1   1  99699  99869      1            HRA1
2   1 147596 151168     -1         YAL001C
3   1 143709 147533      1         YAL002W
4   1 142176 142368      1         YAL003W
5   1 142471 143162      1         YAL003W
6   1 139505 140761     -1         YAL005C
```

In this step we loose all of the additional columns of the `yAnno` object.

4 Statistical Analysis

We can see how "good" the replicates are by assessing whether it fits the Poisson model of constant gene expression across lanes with variable sequencing effort.

```
> groups <- gsub("_[0-9]_f", "", colnames(geneCounts))
> groups
> plot(regionGoodnessOfFit(geneCounts, groups), chisq = TRUE)
```

5 Working with Priming Weights

In a recent publication [3] we describe how the use of random priming for Illumina RNA-Seq impacts the nucleotide content of the reads and we describe a method for alleviating this bias.

The method associates a weight with each read and instead of counting the number of reads in a given genomic interval, the weights in the interval are summed. Because of this, the use of weights happen at the data import step.

Since our example data was generated using random priming, we illustrate the methodology. The first step is to compute the weights using the function `computePrimingWeights` and an *AlignedRead* object. Next, the weights are associated with each read using `addPrimingWeights`. Once reads have an associated weight, the `importFromAlignedReads` function uses these. Because of the need to compute the weights, for now, it is not possible to have `importFromAlignedReads` work directly on filenames.

We start with the `yeastAligned` object which was simply a list of *AlignedRead*, generated using an `lapply` on a vector of filenames.

In this case, we have around 410.000-430.000 reads per sample, with each read having a length of only 26 bases. In order to compute the priming weights we need to assess the k-mer distribution at the end of the reads. In [3], the end is based on reads having at least 35 bases, so we need to modify this. We also makes the weights a bit shorter (as described in Hansen et al., this does not change the effect much)

```
> weightsList <- lapply(yeastAligned, computePrimingWeights,
+                       unbiasedIndex = 20:21, weightsLength = 6L)
> sapply(weightsList, summary)
```

```
           mut_1_f    mut_2_f     wt_1_f     wt_2_f
Min.     0.1377953  0.1415270  0.1725490  0.2047833
1st Qu.  0.7466197  0.7424146  0.7592072  0.7579590
Median   1.1705765  1.1588258  1.1764706  1.1666667
Mean     1.6157932  1.6157863  1.6163375  1.6028931
3rd Qu.  1.9054418  1.9224849  1.9350613  1.9221417
Max.    26.6666667 24.4000000 18.4285714 22.6666667
```

We will continue with a separate set of weights for each lane.

```
> yeastAligned2 <- mapply(addPrimingWeights, yeastAligned, weightsList)
> alignData(yeastAligned2[[1]])
```

```
An object of class 'AlignedDataFrame'
  readName: 1 2 ... 423318 (423318 total)
  varLabels: similar mismatch weights
  varMetadata: labelDescription
```

```
> head(alignData(yeastAligned2[[1]])$weights)
```

```
[1] 0.3824834 0.3925501 0.7834928 0.8106667 1.0075188 1.1889401
```

Now the weights have been added the *AlignedRead* objects. Once this has happened, `importFromAlignedReads` will use them.

```
> eData2 <- importFromAlignedReads(yeastAligned2, chrMap = chrMap,
+                                  dbFilename = "my.db", tablename = "weights")
```

We can now easily get the re-weighted gene level counts as usual.

```
> reweightedCounts <- summarizeByAnnotation(eData2, yAnno, ignoreStrand = TRUE,
+                                           groupBy = "ensembl_gene_id")
> head(reweightedCounts)
```

```
           mut_1_f    mut_2_f    wt_1_f    wt_2_f
15S_rRNA   0.00000   0.000000  0.000000  0.000000
21S_rRNA   0.00000   0.000000  0.000000  0.000000
HRA1       5.04497   9.707854  2.012202 12.249379
LSR1     333.17849 367.034724 47.275727 50.420965
NME1     184.64176 174.321823  5.525353  6.791041
Q0010      0.00000   0.000000  0.000000  0.000000
```

SessionInfo

- R version 3.4.2 Patched (2017-10-07 r73498), `x86_64-w64-mingw32`
- Locale: `LC_COLLATE=C`, `LC_CTYPE=English_United States.1252`, `LC_MONETARY=English_United States.1252`, `LC_NUMERIC=C`, `LC_TIME=English_United States.1252`
- Running under: `Windows Server 2012 R2 x64 (build 9600)`
- Matrix products: default
- Base packages: base, datasets, grDevices, graphics, grid, methods, parallel, stats, stats4, utils
- Other packages: Biobase 2.38.0, BiocGenerics 0.24.0, BiocParallel 1.12.0, Biostrings 2.46.0, DBI 0.7, DelayedArray 0.4.0, GenomeGraphs 1.38.0, GenomeInfoDb 1.14.0, GenomicAlignments 1.14.0, GenomicRanges 1.30.0, Genominator 1.32.0, IRanges 2.12.0, RSQLite 2.0, Rsamtools 1.30.0, S4Vectors 0.16.0, ShortRead 1.36.0, SummarizedExperiment 1.8.0, XVector 0.18.0, biomaRt 2.34.0, matrixStats 0.52.2, yeastRNASeq 0.15.0
- Loaded via a namespace (and not attached): AnnotationDbi 1.40.0, GenomeInfoDbData 0.99.1, Matrix 1.2-11, R6 2.2.2, RColorBrewer 1.1-2, RCurl 1.95-4.8, Rcpp 0.12.13, XML 3.98-1.9, assertthat 0.2.0, bit 1.1-12, bit64 0.9-7, bitops 1.0-6, blob 1.1.0, compiler 3.4.2, digest 0.6.12, hwriter 1.3.2, lattice 0.20-35, latticeExtra 0.6-28, magrittr 1.5, memoise 1.1.0, pkgconfig 2.0.1, prettyunits 1.0.2, progress 1.1.2, rlang 0.1.2, stringi 1.1.5, stringr 1.2.0, tibble 1.3.4, tools 3.4.2, zlibbioc 1.24.0

References

[1] Lee,A., Hansen,K.D., Bullard,J., Dudoit,S. and Sherlock,G. (2008) Novel Low Abundance and Transient RNAs in Yeast Revealed by Tiling Microarrays and Ultra High-Throughput Sequencing Are Not Conserved Across Closely Related Yeast Species. *PLoS Genet*, **4**e1000299.

[2] Bullard,J.H., Purdom,E.A., Hansen,K.D., and Dudoit,S. (2010) Evaluation of statistical methods for normalization and differential expression in mRNA-Seq experiments. *BMC Bioinformatics*, **11**, 94.

[3] Hansen,K.D., Brenner,S.E. and Dudoit,S. (2010) Biases in Illumina transcriptome sequencing caused by random hexamer priming. *Nucleic Acids Res*, doi: 10.1093/nar/gkq224