Working with the ShortRead Package

James Bullard, Kasper Daniel Hansen

January 1, 2010

In this document we show how to use the Genominator package with ShortRead. In this document, we will demonstrate some analysis – maybe even some useful first steps when analyzing RNA-Seq data.

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

We will use some data sets from the yeastRNASeq data package. Lets first see what this package contains:

```
> data(package = "yeastRNASeq")[[3]][, c("Item", "Title")]
```

```
     Item
[1,] "geneLevelData"
[2,] "yeastAligned"
[3,] "yeastAnno"
     Title
[1,] "Yeast gene-level counts from: Lee et al. PloS Genetics 2008."
[2,] "AlignedRead list"
[3,] "Yeast Annotation"
```

At first, we will focus on the AlignedRead list. This object is a list of 4 `AlignedRead`'s, each representing the alignment of .5 million reads. The 4 elements of the list corresponds to 2 samples (mut, wt) each sequenced on 2 lanes.

```
> data("yeastAligned")
> names(yeastAligned)
```

```
[1] "mut_1_f" "mut_2_f" "wt_1_f"  "wt_2_f"
```

```
> 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
```

The first step is to make a database:

```
> eData <- importFromAlignedReads(yeastAligned, chrMap = levels(chromosome(yeastAligned[[1
+     filename = "my.db", tablename = "raw", overwrite = TRUE)
> head(eData)

   chr location strand mut_1_f mut_2_f wt_1_f wt_2_f
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     4101     -1      NA      NA     NA      1
5    1     4242      1       1      NA     NA     NA
6    1     4271     -1       1      NA     NA     NA
7    1     4400      1      NA      NA     NA      1
8    1     4428      1       1      NA     NA     NA
9    1     4447      1      NA      NA     NA      1
10   1     4553     -1      NA       1     NA     NA
```

`importFromAlignedReads` takes exactly such a (named) list of `AlignedReads` and aggregates each experiment and finally joins them together.

Certainly, the most cryptic portion of this call is the chromosome map (the `chrMap` argument). The Genominator package requires that all chromosomes be stored as integer values. Therefore, we must to convert the "chromosome names" to integers. We do this by specifying a rule described as a named vector of integer values, like

```
> levels(chromosome(yeastAligned[[1]]))

 [1] "Scchr01" "Scchr02" "Scchr03" "Scchr04" "Scchr05" "Scchr06"
 [7] "Scchr07" "Scchr08" "Scchr09" "Scchr10" "Scchr11" "Scchr12"
[13] "Scchr13" "Scchr14" "Scchr15" "Scchr16" "Scmito"
```

Sometimes the call to `levels` is not enough and we need to do some reordering and perhaps the addition of extra chromosomes.

The result is an ExpData object which has been aggregated over, i.e. each position in the genome which was represented by > 0 reads at its 5' most end will have that number of reads. Note how we deal with reads mapping to the reverse strand of the genome (by using the 5' end of the read as its location).

We can now take advantage of all of the functionality of the Genominator package, using (parts of) the yeast annotation from SGD.

```
> data(yeastAnno)
> head(yeastAnno)

  ensembl_gene_id chromosome_name start_position end_position strand
1         YHR055C            VIII         214535       214720     -1
2         YPR161C             XVI         864445       866418     -1
3         YOL138C              XV          61325        65350     -1
4         YDR395W              IV        1263316      1266150      1
```

```
5         YGR129W             VII         750405       751052      1
6         YPR165W             XVI         875364       875993      1
    gene_biotype
1 protein_coding
2 protein_coding
3 protein_coding
4 protein_coding
5 protein_coding
6 protein_coding
```

Once again, we have to deal with the fact that people like to represent chromosomes in different ways. In this case, SGD represents chromosomes with Roman numerals. Additionally, we prepare the annotation object to be consumable by the Genominator functions by making sure that we have columns `start`, `end` present and that `strand` takes values in $\{-1, 0, 1\}$.

```
> yeastAnno$chr <- match(yeastAnno$chr, c(as.character(as.roman(1:16)),
+     "MT", "2-micron"))
> yeastAnno$start <- yeastAnno$start_position
> yeastAnno$end <- yeastAnno$end_position
> rownames(yeastAnno) <- yeastAnno$ensembl
> yeastAnno <- yeastAnno[, c("chr", "start", "end", "strand",
+     "gene_biotype")]
> head(yeastAnno)
```

```
        chr   start     end strand   gene_biotype
YHR055C   8  214535  214720     -1 protein_coding
YPR161C  16  864445  866418     -1 protein_coding
YOL138C  15   61325   65350     -1 protein_coding
YDR395W   4 1263316 1266150      1 protein_coding
YGR129W   7  750405  751052      1 protein_coding
YPR165W  16  875364  875993      1 protein_coding
```

With this amount of processing we are now able to do some high level analysis. First we compute, for each annotated region, the number of reads hitting that region. Note that we may double count reads, if two regions overlap (which they often do in yeast).

```
> geneCounts <- summarizeByAnnotation(eData, yeastAnno,
+     ignoreStrand = TRUE)
> head(geneCounts)
```

```
        mut_1_f mut_2_f wt_1_f wt_2_f
YHR055C       0       0      0      0
YPR161C      38      39     35     34
YOL138C      31      33     40     26
YDR395W      55      52     47     47
YGR129W      29      26      5      5
YPR165W     189     180    151    180
```

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.

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