Practical 3: A ChIP-Seq Workflow

Martin Morgan, Marc Carlson, and Patrick Aboyoun

Fred Hutchinson Cancer Research Center, Seattle, WA 98008

7-9 June, 2010

1   Introduction

These exercises focus on a ChIP-seq work flow described in a vignette in the *GenomicRanges* package.

```
> library("GenomicRanges")
```

We use a subset of the ChIP-seq data for origin recognition complex (ORC) binding sites in *Saccharomyces cerevisiae* from the paper *Conserved nucleosome positioning defines replication origins*, Eaton et al. (PMID 20351051). The subset consists of all the MAQ alignments to chromosome XIV for two replicates of ORC ChIP-seq data. The subset is contained in the *EatonEtAlChIPseq* package, and we will use a helper function available in the *EMBL2010* package

```
> library("EatonEtAlChIPseq")
> library("EMBL2010")
```

2   Input and Quality Assessment

We start with looking at the alignments, which were created using MAQ (Li et al. 2008) with a maximum mismatch of 3 bases and a minimum Phred quality score of 35. The data contained in the *EatonEtAlChIPseq* package were obtained and extracted from GEO files GSM424494_wt_G2_orc_chip_rep1.mapview.txt.gz and GSM424494_wt_G2_orc_chip_rep2.mapview.txt.gz.

```
> extdataDir <- system.file("extdata", package="EatonEtAlChIPseq")
> fls <- list.files(extdataDir, ".*gz$", full=TRUE)
> basename(fls)

[1] "GSM424494_wt_G2_orc_chip_rep1_S288C_14.mapview.txt.gz"
[2] "GSM424494_wt_G2_orc_chip_rep2_S288C_14.mapview.txt.gz"
```

2.1   Input

These files can be read in using `readAligned` from the *ShortRead* package, with `type="MAQMapview"`. Here we read in the first replicate.

```
> orcAlignsRep1 <- readAligned(fls[[1]], type = "MAQMapview")
> orcAlignsRep1

class: AlignedRead
length: 478774 reads; width: 39 cycles
chromosome: S288C_14 S288C_14 ... S288C_14 S288C_14
position: 2 4 ... 784295 784295
strand: + - ... + +
alignQuality: IntegerQuality
alignData varLabels: nMismatchBestHit mismatchQuality nExactMatch24 nOneMismatch24
```

A necessary step is to ensure that the `chromosome` naming of `orcAlignsRep1` is consistent with the conventions used in steps down-stream; here we want to change `S288C_14` to `chrXIV`. Using the tools for that from the *ShortRead* package, we do this as follows.

```
> chromosome <- chromosome(orcAlignsRep1)
> levels(chromosome) <- "chrXIV"
> orcAlignsRep1 <- renew(orcAlignsRep1, chromosome=chromosome)
```

Similar actions can be used to read in and recode the second replicate.

A second correction is also required, reflecting an idiosyncracy in data handling. *ShortRead* expects all DNA sequence and quality scores to read 5' to 3' from left to right. On the other hand, in `orcAlignsRep1`, reads on the minus strand are represented by their reverse complement, as they would appear when aligned to the reference sequence. The utility function `renewEatonEtAl`, from the *EMBL2010* package will do the necessary adjustment for us. It reads on the minus strand to conform to *ShortRead* expectations. Here is its code:

```
> renewEatonEtAl

function (aln)
{
    chr <- chromosome(aln)
    levels(chr) <- "chrXIV"
    idx <- strand(aln) == "-"
    qual <- quality(quality(aln))
    qual[idx] <- reverse(qual[idx])
    seq <- sread(aln)
    seq[idx] <- reverseComplement(seq[idx])
    renew(aln, sread = seq, quality = FastqQuality(qual), chromosome = chr)
}
<environment: namespace:EMBL2010>
```

This step is not normally required.

2.2 Quality Assessment

As an exercise, run the `qa` and `report` functions from the *ShortRead* package on the two replicates. A wrinkle is that the reads of the Eaton et al. dataset are represented in an older file format (MAQMapview) that is not supported by `qa`; so, here we perform the quality assessment on the aligned reads only, giving the `qa` a list of *AlignedRead* objects.

```
> data(orcAlignsRep2)
> qa <- qa(list(rep1=renewEatonEtAl(orcAlignsRep1),
+                rep2=renewEatonEtAl(orcAlignsRep2)))
> rpt <- report(qa)
> browseURL(rpt)
```

Some caution in interpretation of results is required, as the reads have already been processed to some extent (e.g., low quality reads removed). Nonetheless, there are several unusual features of the reads. Discuss these with your neighbors.

3 Pre-Processing: Filtering Peaks

For illustration purposes we will focus on the first replicate.

```
> orcAlignsRep1

class: AlignedRead
length: 478774 reads; width: 39 cycles
chromosome: chrXIV chrXIV ... chrXIV chrXIV
position: 2 4 ... 784295 784295
strand: + - ... + +
alignQuality: IntegerQuality
alignData varLabels: nMismatchBestHit mismatchQuality nExactMatch24 nOneMismatch24
```

In this subsection we will demonstrate how to perform three filtering operations on alignments produced by the MAQ software through the following restrictions:

- Number of mismatches in alignment must be ≤ 3 (guideline specified in paper)

- No duplicates of {chromosome, strand, position} combinations (PCR bias correction)

- An alignment on one strand must have a plausible alignment on the complementary strand ("symmetry" restriction)

The first two restrictions can be implemented using functionality from the *ShortRead* package, while the last one can be performed using operations within the *GenomicRanges* package. Filters similar to these are implemented in many peak calling algorithms.

In the previous section we loaded the `orcAlignsRep1` object, an instance of the *ShortRead* class *AlignedRead*. This object contains information on the characteristics of the read as well as its alignment to a reference genome, including information on the number of mismatches for the best alignment. Using functionality from the *ShortRead* package we can perform the first two filtering operations, which result in a subset that is roughly 18% of the size of the original MAQ alignment file.

```
> subsetRep1 <-
+   orcAlignsRep1[alignData(orcAlignsRep1)[["nMismatchBestHit"]] <= 3]
> length(subsetRep1) / length(orcAlignsRep1)
```

```
[1] 0.96655

> subsetRep1 <- subsetRep1[occurrenceFilter(withSread=FALSE)(subsetRep1)]
> length(subsetRep1) / length(orcAlignsRep1)

[1] 0.180722

> subsetRep1

class: AlignedRead
length: 86525 reads; width: 39 cycles
chromosome: chrXIV chrXIV ... chrXIV chrXIV
position: 2 5 ... 784294 784295
strand: + + ... - +
alignQuality: IntegerQuality
alignData varLabels: nMismatchBestHit mismatchQuality nExactMatch24 nOneMismatch24
```

The last filtering criterion, a "symmetry" filter, requires an understanding of the interval spans for the alignments rather than just the "leftmost" alignment location, i.e. start location on the positive strand or end location on the negative strand, represented in the *AlignedRead* class. As such we will coerce the alignment subset contained in `subsetRep1` to a *GRanges* object using a `coerce` method from the [ShortRead] package.

```
> rangesRep1 <- as(subsetRep1, "GRanges")
> head(rangesRep1, 3)

GRanges with 3 ranges and 4 elementMetadata values
    seqnames     ranges strand | nMismatchBestHit
       <Rle> <IRanges>  <Rle> |        <integer>
[1]   chrXIV   [2, 40]      + |                0
[2]   chrXIV   [5, 43]      + |                0
[3]   chrXIV   [6, 44]      + |                1
    mismatchQuality nExactMatch24 nOneMismatch24
          <integer>     <integer>      <integer>
[1]                0             5              0
[2]                0             5              0
[3]                4             6              0

seqlengths
 chrXIV
     NA
```

AlignedRead objects lack information on chromosome length, so we will add it to the new `rangesRep1` object.

```
> seqlengths(rangesRep1) <- 784333
```

For our "pseudo" paired-read filter, we will use the authors' estimate of the mean fragment length (around 150 base pairs). In particular, we will construct a filter where each alignment on the plus strand must have a corresponding

alignment somewhere within [100, 200] bp downstream on the minus strand and vice versa with those alignments on the minus strand.

This filtering process can be achieved through an interval overlap operation between the starts of the alignments on the minus strand and the projected end of the alignments on the plus strand, where the former can be derived by the code

```
> negRangesRep1 <- rangesRep1[strand(rangesRep1) == "-"]
> negStartsRep1 <- resize(negRangesRep1, 1)
```

and the latter by the code

```
> posRangesRep1 <- rangesRep1[strand(rangesRep1) == "+"]
> posEndsRep1 <- shift(posRangesRep1, 99)
> posEndsRep1 <- resize(posEndsRep1, 100)
> strand(posEndsRep1) <- "-"
```

The results of the interval overlap are shown below. This filter flagged roughly 4.5% of the remaining alignments for removal, resulting in keeping 17.3% of the original set of alignments when including the results of the first two filtering steps.

```
> strandMatching <- findOverlaps(negStartsRep1, posEndsRep1)
> posKeep <- unique(subjectHits(strandMatching))
> negKeep <- unique(queryHits(strandMatching))
> length(posKeep) / length(posEndsRep1)

[1] 0.9559064

> length(negKeep) / length(negStartsRep1)

[1] 0.9554474

> (length(posKeep) + length(negKeep)) / length(orcAlignsRep1)

[1] 0.1727120

> posFilteredRangesRep1 <- posRangesRep1[posKeep]
> negFilteredRangesRep1 <- negRangesRep1[negKeep]
```

4   Finding peaks

Once the alignments have been filtered, they can be aggregated into coverage vectors. Many peak calling algorithms use a point estimate for the average fragment size of each read. We suppose that the size of the reads are uniformly distributed over [100, 200] base pairs, although more elaborate distributions are readily modeled. We impelement our model by weighting the coverage vector of the original reads with a function that gives full weight to positions 100 bp upstream, and then linearly decreasing weights for the next 100 bp. For the runwtsum function from *IRanges*, these weights for the positive and negative strands are expressed by the vectors:

5

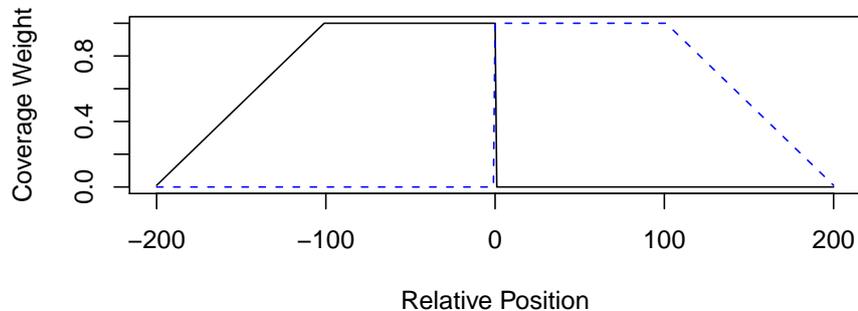Figure 1: Coverage weights for positive strand weighted sums.

```
> posWeights <- c(seq(0.01, 1, length = 100), rep(c(1, 0), c(101, 200)))
> negWeights <- rev(posWeights)
> plot(-200:200, posWeights, xlab = "Relative Position",
+       ylab = "Coverage Weight", type = "l")
> lines(-200:200, negWeights, lty=2, col="blue")
```

The first step in constructing this coverage vector is to tabulate the alignments by their start positions on both the positive and negative strand. This is done with the `resize` function, whose second argument is the width (in this case, 1) of the resulting fragment. We will use the `coverage` function on these start values, which will produce *RleList* representations of the coverage vectors.

```
> posStartsCoverRep1 <- coverage(resize(posFilteredRangesRep1, 1))
> negStartsCoverRep1 <- coverage(resize(negFilteredRangesRep1, 1))
```

The second step in the process is aggregating upstream alignments using the `posWeights` and `negWeights` objects defined above.

```
> posExtCoverRep1 <-
+   round(runwtsum(posStartsCoverRep1, k = 401, wt = posWeights,
+                 endrule = "constant"))
> negExtCoverRep1 <-
+   round(runwtsum(negStartsCoverRep1, k = 401, wt = negWeights,
+                 endrule = "constant"))
```

Before we proceeded any further, we will define two plot functions for visualizing coverage vectors: `plotCoverage` for displaying a single coverage vector and `plotStrandedCoverage` for displaying back-to-back coverage vectors for dual-stranded data.

```
> plotCoverage <-
+ function(x, xlab = "Position", ylab = "Coverage",...)
```
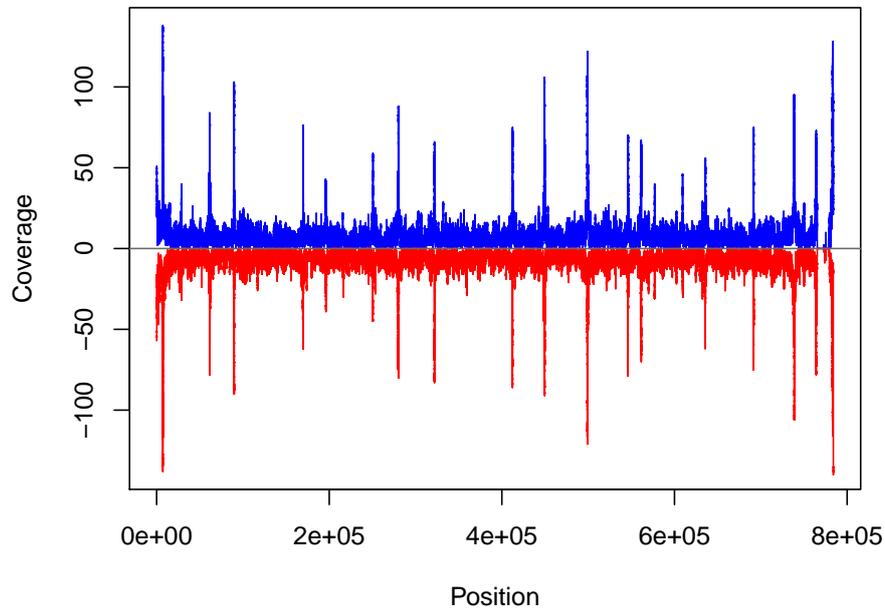
Figure 2: Plot of coverage across chromosome XIV.

```
+ {
+     plot(c(start(x), length(x)), c(runValue(x), tail(runValue(x), 1)),
+          type = "s", col = "blue", xlab = xlab, ylab = ylab, ...)
+ }
> plotStrandedCoverage <-
+ function(positive, negative, xlab = "Position", ylab = "Coverage",...)
+ {
+     ylim <- min(max(positive), max(negative)) * c(-1, 1)
+     plotCoverage(positive, ylim = ylim, ...)
+     lines(c(start(negative), length(negative)),
+           - c(runValue(negative), tail(runValue(negative), 1)),
+           type = "s", col = "red")
+     abline(h = 0, col = "dimgray")
+ }
```

The coverage across chromosome XIV of the filtered alignements is shown in Figure 2. In general this plot shows a near mirror image of coverage vectors between the positive and negative strand.

```
> plotStrandedCoverage(posExtCoverRep1[[1]], negExtCoverRep1[[1]])
```

To reduce these measures to a single dimension, we will be conservative and choose the smallest value between the positive and negative strand coverage

vectors using the `pmin` method for *RleList* objects.

The experimental design of Eaton et al. does not include a 'control' lane; such lanes are commonly included in transcription factor and other ChIP-seq experiments. Many software packages implement more elaborate approaches to modeling peaks, for example the PeakSeq, MACS, SBP, *BayesPeak*, and SWEMBL packages; this software is primarily useful for transcription factor style, but not nucleosome, analysis. A review of alternatives is avialable.

```
> combExtCoverRep1 <- pmin(posExtCoverRep1, negExtCoverRep1)
> quantile(combExtCoverRep1, c(0.5, 0.9, 0.95))

      chrXIV
50%        4
90%       10
95%       14
```

We now can call peaks off the combined coverage object `combExtCoverRep1`. Since the median height for the combined coverage on chromosome XIV is 4, we can limit our attention to areas on the chromosome with coverage ≥ 5 using the `slice` function. From there we can derive a heuristic for a meaningful peaks as those achieving a maximum height ≥ 28, which selects 22 peaks.

```
> peaksRep1 <- slice(combExtCoverRep1, lower = 5)
> peakMaxsRep1 <- viewMaxs(peaksRep1)
> tail(sort(peakMaxsRep1[[1]]), 30)

 [1]  22  22  22  23  23  23  24  25  28  29  29  32  41  51
[15]  51  58  61  64  66  69  73  73  74  75  83  89  90 114
[29] 116 137

> peaksRep1 <- peaksRep1[peakMaxsRep1 >= 28]
> peakRangesRep1 <-
+   GRanges("chrXIV", as(peaksRep1[[1]], "IRanges"),
+           seqlengths = seqlengths(rangesRep1))
> length(peakRangesRep1)

[1] 22
```

We can now compare the significant peaks we selected with those selected by the authors. Using interval comparison tools, we see there is general agreement between our peaks and those of the authors.

```
> data(orcPeaksRep1)
> countOverlaps(orcPeaksRep1, peakRangesRep1)

 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

> countOverlaps(peakRangesRep1, orcPeaksRep1)

 [1] 0 2 0 1 1 1 1 1 1 1 1 1 2 1 1 0 0 1 1 1 1 1
```

Eaton et al. continue their analysis in a number of interesting directions (e.g., motif characterization, nucleosome positioning), and it is left as an open exercise to explore *R / Bioconductor* solutions to these problems.

5 Analysis of designed experiments

The approach outlined above does not make use of the second experimental replicate of Eaton et al., and in general analysis of designed experiments is still relatively unexplored – studies to date often include technical replicates, but use this data by pooling across replicates (lanes) prior to peak calling. Packages such as *DESeq* may be appropriate for analysis of designed experiments once peaks have been identified. This approach is likely to be appropriate when the assay targets clearly separated or previously identified binding sites; it would not be appropriate when the assay targets broad or poorly defined peaks. This represents an interesting area for further development.