Combine_tutorial

Anna Qualieri

7 May 2017

Contents

1 GenomicRanges: Granges and GRangesList

Let's start by loading the GenomicRanges package via *biocLite*:

```
> source("https://bioconductor.org/biocLite.R")
> biocLite("GenomicRanges")
```

Load the packages necessary for this analysis into R. You can use the *install.packages(package = "mypackage")* for every CRAN package below than is not already installed in your R session or *biocLite* for *Bioconductor* packages.

```
> library(GenomicRanges)
> library(ggplot2)
```

1.1 *GRanges*

Now that we are familiar with the basic structure and functionality of *GRanges* objects we are going to explore other ways of manipulating their information as well as ways to compare different *GRanges* objects.

Imagine you have done a ChIP-Seq experiment on *Sample 1* and your output is a set of ranges. Let's start by manually creating this very simple *GRanges* objects with regions only on *chr1* and *chr2*.

```
> # Sample 1
> gr_S1 <- GRanges(seqnames = Rle(c("chr1", "chr2"), c(3, 2)), ranges = IRanges(start = c(5,
+     8, 20, 8, 18), end = c(11, 15, 26, 16, 21), names = c(paste("Peak_", 1:5,
+     sep = ""))), strand = Rle(strand(c("*")), c(5)), peak_coverage = rbinom(n = 5,
+     size = 10, prob = 0.6))
>
> gr_S1
```

```
## GRanges object with 5 ranges and 1 metadata column:
##          seqnames    ranges strand | peak_coverage
##             <Rle> <IRanges>  <Rle> |     <integer>
##   Peak_1     chr1  [ 5, 11]      * |             6
##   Peak_2     chr1  [ 8, 15]      * |             8
##   Peak_3     chr1  [20, 26]      * |             9
##   Peak_4     chr2  [ 8, 16]      * |             5
##   Peak_5     chr2  [18, 21]      * |             7
##   -------
##   seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

A very common analysis to perform is to evaluate to what extent and where your ChIP-Seq peaks overlap with some **features**, such as genes, exons, other ChIP-Seq peaks, etc... As an example we create a simple gene annotation to use with the ranges created above.

```
> # Gene annotation
> genes <- GRanges(seqnames = Rle(c("chr1", "chr2"), c(2, 2)), ranges = IRanges(start = c(7,
+     17, 7, 23), end = c(15, 23, 14, 26), names = c(paste("Gene_", 1:4, sep = ""))),
+     strand = Rle(strand(c("+", "+")), c(2, 2)))
>
> genes
```

```
## GRanges object with 4 ranges and 0 metadata columns:
##          seqnames    ranges strand
##             <Rle> <IRanges>  <Rle>
##   Gene_1     chr1  [ 7, 15]      +
##   Gene_2     chr1  [17, 23]      +
##   Gene_3     chr2  [ 7, 14]      +
##   Gene_4     chr2  [23, 26]      +
##   -------
##   seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

Figure 1 is a simple way of plotting ranges stored into *Granges* object using *geom_rect* from *ggplot2*. Even though there are available more advanced ways of plotting genomic data, this simple strategy is sufficient for the purpose of this tutorial. This is an example of how any *GRanges* object can be converted to a *data.frame* via *data.frame(myGranges)*.

```
> gr <- data.frame(rbind(data.frame(gr_S1), cbind(data.frame(genes), peak_coverage = 5)))
> gr$rangeID <- c(names(gr_S1), names(genes))
> gr$Sample <- c(rep("Peaks", length(gr_S1)), rep("Genes", length(genes)))
> ggplot(data = gr, aes(xmin = start, xmax = end, ymin = 0, ymax = peak_coverage)) +
+     geom_rect(aes(fill = rangeID), alpha = 0.4) + facet_wrap(~Sample + seqnames)
```

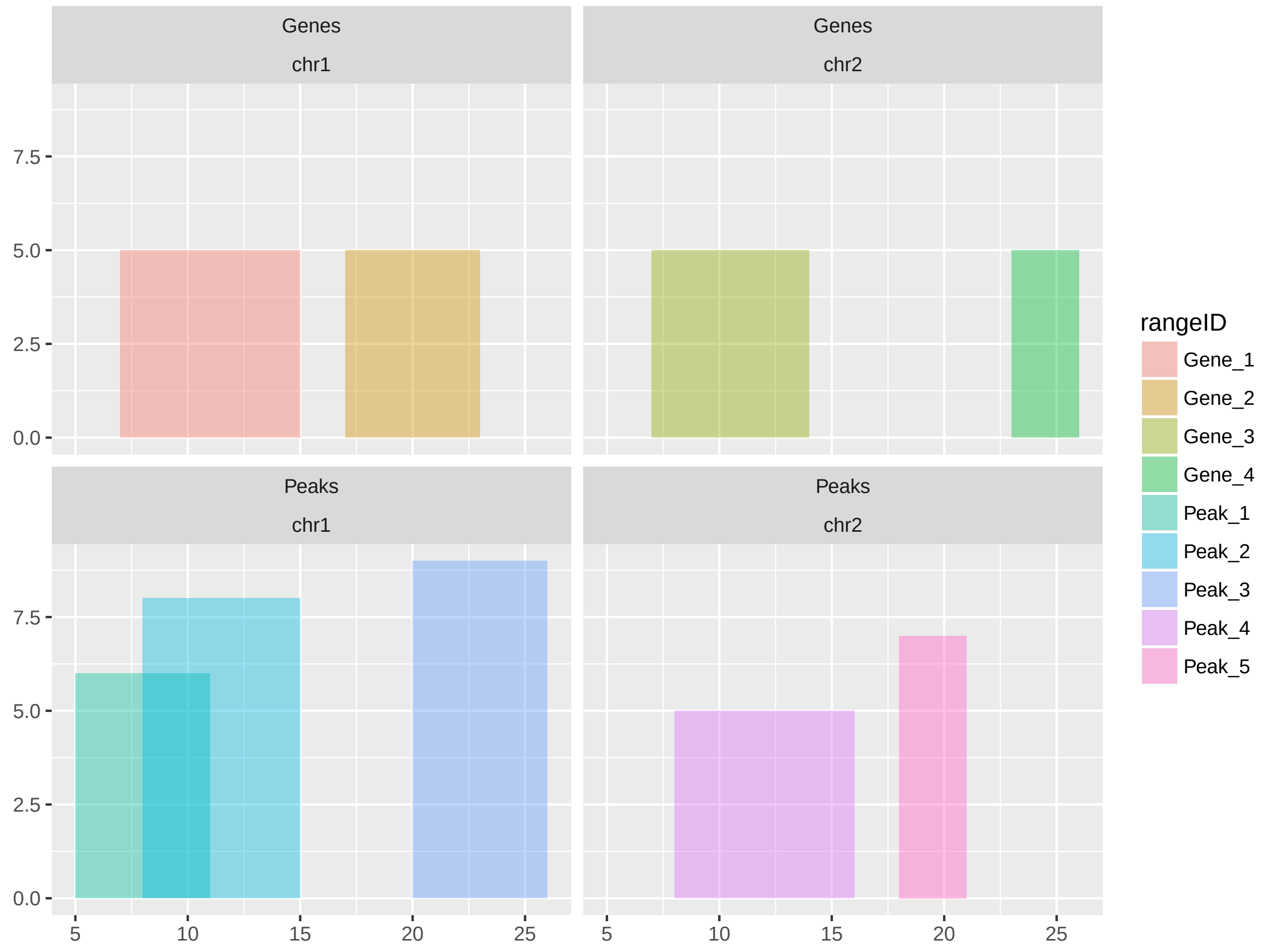

Figure 1: Graphic representation of the GRanges objects created above.

1.2 Find overlaps between two *GRanges* objects

From Figure 1 one notices that the two ranges in *chr1* from *Sample 1* are overlapping. This happens in ChIP-Seq experiments for example when there is a very broad peak and several peaks are called in place of one. Usually, it is helpful to simplify these situations with the aim of reducing a *GRanges* object to its minimal set of ranges, merging all such overlapping regions. This can be easily achieved with the *reduce()* function:

```
> gr_S1_reduced <- reduce(gr_S1)
> gr_S1_reduced
```

```
## GRanges object with 4 ranges and 0 metadata columns:
##       seqnames    ranges strand
##          <Rle> <IRanges>  <Rle>
##   [1]     chr1  [ 5, 15]      *
##   [2]     chr1  [20, 26]      *
##   [3]     chr2  [ 8, 16]      *
##   [4]     chr2  [18, 21]      *
##   -------
##   seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

```
> length(gr_S1)
```

```
## [1] 5
```

```
> length(gr_S1_reduced)
```

```
## [1] 4
```

One can also decide to keep every range distinct and evaluate the overlap for each of them. For this analysis we will keep the overlapping ranges as disticnt regions.

1.2.1 *findOverlaps()*

Now, *findOverlaps()* can be used to detect overlap between the ChIP-Seq ranges and the gene annotation. *findOverlap()* by default looks for overlaps starting from 1bp between a *query* and a *subject* and it does not allow any gap between the overlapping ranges (arguments *minoverlap* and *maxgap* are 1 and 0 respectively by default).

```
> overlaps <- findOverlaps(gr_S1, genes)
> overlaps
```

```
## Hits object with 4 hits and 0 metadata columns:
##       queryHits subjectHits
##       <integer>   <integer>
##   [1]         1           1
##   [2]         2           1
##   [3]         3           2
##   [4]         4           3
##   -------
##   queryLength: 5 / subjectLength: 4
```

Below is an example of what happens if we change the *maxgap* argument to 5, allowing 5bp to be present between ranges:

```
> overlaps <- findOverlaps(gr_S1, genes, maxgap = 5)
> overlaps
```

```
## Hits object with 8 hits and 0 metadata columns:
##       queryHits subjectHits
##       <integer>   <integer>
##   [1]         1           1
##   [2]         2           1
##   [3]         2           2
##   [4]         3           1
##   [5]         3           2
##   [6]         4           3
##   [7]         5           3
##   [8]         5           4
##   -------
##   queryLength: 5 / subjectLength: 4
```

1.2.2 *countOverlaps()*

The example used here is very simple and it straightforward to see how many ranges are overlapping with genes and viceversa. However, for more complex experiments the *countOverlaps()* function is a very useful tool to get a quick summary of the overlaps for every range.

```
> N_overlaps <- countOverlaps(gr_S1, genes)
> N_overlaps
```

```
## Peak_1 Peak_2 Peak_3 Peak_4 Peak_5
##      1      1      1      1      0
```

1.3 Nearest-methods in *GenomicRanges*

1.3.1 *nearest()*

Will return a vector of indeces referring to the nearest neighbour *subject* for every range in *x*.

By default if one range overlaps with multiple genes then one overlap will be chosen at random:

```
> GenomicRanges::nearest(x = gr_S1, subject = genes)
```

```
## [1] 1 1 2 3 4
```

Using *select = "all"* all overlaps will be return:

```
> GenomicRanges::nearest(x = gr_S1, subject = genes, select = "all")
```

```
## Hits object with 5 hits and 0 metadata columns:
##       queryHits subjectHits
##       <integer>   <integer>
##   [1]         1           1
##   [2]         2           1
##   [3]         3           2
##   [4]         4           3
##   [5]         5           4
##   -------
##   queryLength: 5 / subjectLength: 4
```

You can also look for the nearest-neighbours within a single set of ranges:

```
> GenomicRanges::nearest(gr_S1)
```

```
## [1] 2 1 2 5 4
```

1.3.2 *distance()*

```
> GenomicRanges::distance(x = gr_S1[1], y = genes)
```

```
## [1]  0  5 NA NA
```

```
> GenomicRanges::distance(x = gr_S1[1:4], y = genes)
```

```
## [1]  0  1 NA  6
```

```
> GenomicRanges::distance(x = gr_S1, y = genes)
```

```
## [1]  0  1 NA  6 NA
```

distance() is a symmetric function which means that it requires x and y to have to have the same lenght and if one is shorter than the other one it will be recycled to match the length of the longest. Also, the distance between two consecutive blocks is 0 not 1 which affects the notion of overlaps. If distance(x, y) == 0 then x and y can be either adjacent or overlapping ranges. For more information about the *distance()* function see ?IRanges::distance.

1.3.3 *distanceToNearest()*

For every range in x it will return the index and the distance to its nearest neighbour in *subject*.

```
> GenomicRanges::distanceToNearest(x = gr_S1, subject = genes)
```

```
## Hits object with 5 hits and 1 metadata column:
##       queryHits subjectHits |  distance
##       <integer>   <integer> | <integer>
##   [1]         1           1 |         0
##   [2]         2           1 |         0
##   [3]         3           2 |         0
##   [4]         4           3 |         0
##   [5]         5           4 |         1
##   -------
##   queryLength: 5 / subjectLength: 4
```

1.4 *GRangesList*

GRangesList are lists of *GRanges* objects. In some instances it makes sense to store several set of ranges under a common parent. For example, when you call peaks on several technical replicates from a ChIP-Seq experiments and you want to store their output in one object. Anotehr example is when you would like to store different transcripts from the same gene under a common object. Below, this concept is illustrated with a simple example:

```
> # Sample 1
> gr_S1 <- GRanges(seqnames = Rle(c("chr1", "chr2"), c(3, 2)), ranges = IRanges(start = c(5,
+     8, 20, 8, 18), end = c(11, 15, 26, 16, 21), names = c(paste("Peak_", 1:5,
+     sep = ""))), strand = Rle(strand(c("*")), c(5)), peak_coverage = rbinom(n = 5,
+     size = 10, prob = 0.6))
>
> gr_S1
```

```
## GRanges object with 5 ranges and 1 metadata column:
##          seqnames    ranges strand | peak_coverage
##             <Rle> <IRanges>  <Rle> |     <integer>
##   Peak_1     chr1  [ 5, 11]      * |             3
##   Peak_2     chr1  [ 8, 15]      * |             6
##   Peak_3     chr1  [20, 26]      * |             5
##   Peak_4     chr2  [ 8, 16]      * |             6
##   Peak_5     chr2  [18, 21]      * |             7
##   -------
##   seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

```
> # Sample 2
> gr_S2 <- GRanges(seqnames = Rle(c("chr2", "chr3"), c(3, 5)), ranges = IRanges(start = c(1:8),
+     width = 10, names = c(paste("Peak_", 1:8, sep = ""))), strand = Rle(strand(c("*")),
+     c(8)), peak_coverage = rbinom(n = 8, size = 10, prob = 0.6))
>
> gr_S2
```

```
## GRanges object with 8 ranges and 1 metadata column:
##          seqnames    ranges strand | peak_coverage
##             <Rle> <IRanges>  <Rle> |     <integer>
##   Peak_1     chr2   [1, 10]      * |             8
##   Peak_2     chr2   [2, 11]      * |             6
##   Peak_3     chr2   [3, 12]      * |             5
##   Peak_4     chr3   [4, 13]      * |             8
##   Peak_5     chr3   [5, 14]      * |             6
##   Peak_6     chr3   [6, 15]      * |             7
##   Peak_7     chr3   [7, 16]      * |             8
##   Peak_8     chr3   [8, 17]      * |             7
##   -------
##   seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

```
> # GRanges List
>
> list_ranges <- GRangesList(Sample1 = gr_S1, Sample2 = gr_S2)
```

Many of the functions learnt for *GRanges* can also be applied to *GRangesList* objects even though the output will have to be interepreted accordingly:

```
> names(list_ranges)
```

```
## [1] "Sample1" "Sample2"
```

```
> length(list_ranges)
```

```
## [1] 2
```

```
> seqnames(list_ranges)
```

```
## RleList of length 2
## $Sample1
## factor-Rle of length 5 with 2 runs
##   Lengths:    3    2
##   Values : chr1 chr2
## Levels(3): chr1 chr2 chr3
##
## $Sample2
## factor-Rle of length 8 with 2 runs
##   Lengths:    3    5
##   Values : chr2 chr3
## Levels(3): chr1 chr2 chr3
```

```
> strand(list_ranges)
```

```
## RleList of length 2
## $Sample1
## factor-Rle of length 5 with 1 run
##   Lengths: 5
##   Values : *
## Levels(3): + - *
##
## $Sample2
## factor-Rle of length 8 with 1 run
##   Lengths: 8
##   Values : *
## Levels(3): + - *
```

```
> ranges(list_ranges)
```

```
## IRangesList of length 2
## $Sample1
## IRanges object with 5 ranges and 0 metadata columns:
##               start       end     width
##           <integer> <integer> <integer>
##   Peak_1          5        11         7
##   Peak_2          8        15         8
##   Peak_3         20        26         7
##   Peak_4          8        16         9
##   Peak_5         18        21         4
```

```
##
## $Sample2
## IRanges object with 8 ranges and 0 metadata columns:
##              start       end     width
##          <integer> <integer> <integer>
##   Peak_1         1        10        10
##   Peak_2         2        11        10
##   Peak_3         3        12        10
##   Peak_4         4        13        10
##   Peak_5         5        14        10
##   Peak_6         6        15        10
##   Peak_7         7        16        10
##   Peak_8         8        17        10
```

```
> start(list_ranges)
```

```
## IntegerList of length 2
## [["Sample1"]] 5 8 20 8 18
## [["Sample2"]] 1 2 3 4 5 6 7 8
```

```
> start(list_ranges)[[1]]
```

```
## [1]  5  8 20  8 18
```

```
> end(list_ranges)
```

```
## IntegerList of length 2
## [["Sample1"]] 11 15 26 16 21
## [["Sample2"]] 10 11 12 13 14 15 16 17
```

```
> width(list_ranges)
```

```
## IntegerList of length 2
## [["Sample1"]] 7 8 7 9 4
## [["Sample2"]] 10 10 10 10 10 10 10 10
```

To get the number of ranges in every object of the list use *elementNROWS*:

```
> elementNROWS(list_ranges)
```

```
## Sample1 Sample2
##       5       8
```

It is also possible to quickly combine all the element of a *GRangesList* into one *GRanges* object:

```
> list_to_granges <- unlist(list_ranges)
> list_to_granges
```

```
## GRanges object with 13 ranges and 1 metadata column:
##                   seqnames    ranges strand | peak_coverage
##                      <Rle> <IRanges>  <Rle> |     <integer>
##   Sample1.Peak_1     chr1  [ 5, 11]      * |             3
##   Sample1.Peak_2     chr1  [ 8, 15]      * |             6
##   Sample1.Peak_3     chr1  [20, 26]      * |             5
##   Sample1.Peak_4     chr2  [ 8, 16]      * |             6
##   Sample1.Peak_5     chr2  [18, 21]      * |             7
##              ...      ...       ...    ... .           ...
##   Sample2.Peak_4     chr3   [4, 13]      * |             8
##   Sample2.Peak_5     chr3   [5, 14]      * |             6
##   Sample2.Peak_6     chr3   [6, 15]      * |             7
##   Sample2.Peak_7     chr3   [7, 16]      * |             8
##   Sample2.Peak_8     chr3   [8, 17]      * |             7
##   -------
##   seqinfo: 3 sequences from an unspecified genome; no seqlengths
```

To append two *GRanges* lists simply use the R concatenate command *c*:

```
> # Sample 3
> gr_S3 <- GRanges(seqnames = Rle(c("chr1", "chr2"), c(3, 2)), ranges = IRanges(start = 20:24,
+     width = 8, names = c(paste("Peak_", 1:5, sep = ""))), strand = Rle(strand(c("*")),
+     c(5)), peak_coverage = rbinom(n = 5, size = 10, prob = 0.6))
>
> gr_S1
```

```
## GRanges object with 5 ranges and 1 metadata column:
##          seqnames    ranges strand | peak_coverage
##             <Rle> <IRanges>  <Rle> |     <integer>
##   Peak_1     chr1  [ 5, 11]      * |             3
##   Peak_2     chr1  [ 8, 15]      * |             6
##   Peak_3     chr1  [20, 26]      * |             5
##   Peak_4     chr2  [ 8, 16]      * |             6
##   Peak_5     chr2  [18, 21]      * |             7
##   -------
##   seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

```
> # Sample 4
> gr_S4 <- GRanges(seqnames = Rle(c("chr2", "chr3"), c(3, 5)), ranges = IRanges(start = 20:27,
+     width = 10, names = c(paste("Peak_", 1:8, sep = ""))), strand = Rle(strand(c("*")),
+     c(8)), peak_coverage = rbinom(n = 8, size = 10, prob = 0.6))
>
> gr_S4
```

```
## GRanges object with 8 ranges and 1 metadata column:
##          seqnames    ranges strand | peak_coverage
##             <Rle> <IRanges>  <Rle> |     <integer>
##   Peak_1     chr2  [20, 29]      * |             6
##   Peak_2     chr2  [21, 30]      * |             5
##   Peak_3     chr2  [22, 31]      * |             5
##   Peak_4     chr3  [23, 32]      * |             6
##   Peak_5     chr3  [24, 33]      * |             6
##   Peak_6     chr3  [25, 34]      * |             4
##   Peak_7     chr3  [26, 35]      * |             5
##   Peak_8     chr3  [27, 36]      * |             6
##   -------
##   seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

```
> # Second GRanges List
> list_ranges2 <- GRangesList(Sample3 = gr_S3, Sample4 = gr_S4)
```

```
> append_lists <- c(list_ranges, list_ranges2)
> head(append_lists)
```

```
## GRangesList object of length 4:
## $Sample1
## GRanges object with 5 ranges and 1 metadata column:
##          seqnames    ranges strand | peak_coverage
##             <Rle> <IRanges>  <Rle> |     <integer>
##   Peak_1     chr1  [ 5, 11]      * |             3
##   Peak_2     chr1  [ 8, 15]      * |             6
##   Peak_3     chr1  [20, 26]      * |             5
##   Peak_4     chr2  [ 8, 16]      * |             6
##   Peak_5     chr2  [18, 21]      * |             7
##
## $Sample2
## GRanges object with 8 ranges and 1 metadata column:
##          seqnames  ranges strand | peak_coverage
##   Peak_1     chr2 [1, 10]      * |             8
##   Peak_2     chr2 [2, 11]      * |             6
##   Peak_3     chr2 [3, 12]      * |             5
##   Peak_4     chr3 [4, 13]      * |             8
##   Peak_5     chr3 [5, 14]      * |             6
##   Peak_6     chr3 [6, 15]      * |             7
##   Peak_7     chr3 [7, 16]      * |             8
##   Peak_8     chr3 [8, 17]      * |             7
##
## $Sample3
## GRanges object with 5 ranges and 1 metadata column:
##          seqnames   ranges strand | peak_coverage
##   Peak_1     chr1 [20, 27]      * |             7
##   Peak_2     chr1 [21, 28]      * |             6
##   Peak_3     chr1 [22, 29]      * |             7
##   Peak_4     chr2 [23, 30]      * |             5
##   Peak_5     chr2 [24, 31]      * |             6
##
## ...
## <1 more element>
## -------
## seqinfo: 3 sequences from an unspecified genome; no seqlengths
```

1.5 Subsetting and looping over *GRanges* list

GRanges objects can be treated

It is often of interest

Anna - More about Ranges and other

Advanced GRanges ; GRangesList findOverlaps(), countOverlaps(), nearest(), distance(), distanceToNearest() GRangesList

Rtracklayer package Import/export bed gff/gtf files into GRanges objects using rtracklayer functions Load bigWig and wig files Use liftOver() to convert genomic coordinates between different genome versions, and where to find the chain files

Rsamtools State some application and connection with GRanges and BSgenome