An Introduction to the GenomicRanges Package

Marc Carlson  Patrick Aboyoun  Hervé Pages

October 19, 2016

Contents

1 Introduction 1

2 GRanges: Genomic Ranges 2
  2.1 Splitting and combining GRanges objects 4
  2.2 Subsetting GRanges objects 4
  2.3 Basic interval operations for GRanges objects 7
  2.4 Interval set operations for GRanges objects 10

3 GRangesList: Groups of Genomic Ranges 11
  3.1 Basic GRangesList accessors 12
  3.2 Combining GRangesList objects 14
  3.3 Basic interval operations for GRangesList objects 14
  3.4 Subsetting GRangesList objects 15
  3.5 Looping over GRangesList objects 17

4 Interval overlaps involving GRanges and GRangesList objects 20

5 Session Information 21

1 Introduction

The GenomicRanges package serves as the foundation for representing genomic locations within the Bioconductor project. In the Bioconductor package hierarchy, it builds upon the IRanges (infrastructure) package and provides support for the BSgenome (infrastructure), Rsamtools (I/O), ShortRead (I/O & QA), rtracklayer (I/O), and GenomicFeatures (infrastructure) packages, and many other Bioconductor packages.

This package lays a foundation for genomic analysis by introducing three classes (GRanges, GPos, and GRangesList), which are used to represent genomic ranges, genomic positions, and groups of genomic ranges. This vignette focuses on the GRanges and GRangesList classes and their associated methods.

The GenomicRanges package is available at bioconductor.org and can be downloaded via biocLite:

```r
> source("https://bioconductor.org/biocLite.R")
> biocLite("GenomicRanges")
> library(GenomicRanges)
```
2 **GRanges**: Genomic Ranges

The *GRanges* class represents a collection of genomic ranges that each have a single start and end location on the genome. It can be used to store the location of genomic features such as contiguous binding sites, transcripts, and exons. These objects can be created by using the *GRanges* constructor function. For example,

```r
> gr <-
+ GRanges(seqnames =
+   Rle(c("chr1", "chr2", "chr1", "chr3"), c(1, 3, 2, 4)),
+   ranges =
+   IRanges(1:10, end = 7:16, names = head(letters, 10)),
+   strand =
+   Rle(strand(c("-", "+", "+", "+", "-")),
+     c(1, 2, 2, 3, 2)),
+   score = 1:10,
+   GC = seq(1, 0, length=10))
> gr

GRanges object with 10 ranges and 2 metadata columns:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
<th>score</th>
<th>GC</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>chr1 [1, 7]</td>
<td>-</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>chr2 [2, 8]</td>
<td>+</td>
<td>2</td>
<td>0.888888888888889</td>
</tr>
<tr>
<td>c</td>
<td>chr2 [3, 9]</td>
<td>+</td>
<td>3</td>
<td>0.777777777777778</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>h</td>
<td>chr3 [8, 14]</td>
<td>+</td>
<td>8</td>
<td>0.222222222222222</td>
</tr>
<tr>
<td>i</td>
<td>chr3 [9, 15]</td>
<td>-</td>
<td>9</td>
<td>0.111111111111111</td>
</tr>
<tr>
<td>j</td>
<td>chr3 [10, 16]</td>
<td>-</td>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>

seqinfo: 3 sequences from an unspecified genome; no seqlengths
```

creates a *GRanges* object with 10 genomic ranges. The output of the *GRanges* `show` method separates the information into a left and right hand region that are separated by `|` symbols. The genomic coordinates (seqnames, ranges, and strand) are located on the left-hand side and the metadata columns (annotation) are located on the right. For this example, the metadata is comprised of `score` and `GC` information, but almost anything can be stored in the metadata portion of a *GRanges* object.

The components of the genomic coordinates within a *GRanges* object can be extracted using the `seqnames`, `ranges`, and `strand` accessor functions.

```r
> seqnames(gr)

factor-Rle of length 10 with 4 runs
  Lengths: 1 3 2 4
  Levels(3): chr1 chr2 chr3

> ranges(gr)

IRanges object with 10 ranges and 0 metadata columns:

<table>
<thead>
<tr>
<th></th>
<th>start</th>
<th>end</th>
<th>width</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>8</td>
<td>7</td>
</tr>
</tbody>
</table>
```
factor-Rle of length 10 with 5 runs
Lengths: 1 2 2 3 2
Values : - + * + -
Levels(3): + - *

Stored annotations for these coordinates can be extracted as a DataFrame object using the mcols accessor.

> mcols(gr)

DataFrame with 10 rows and 2 columns

<table>
<thead>
<tr>
<th>score</th>
<th>GC</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;integer&gt;</td>
<td>&lt;numeric&gt;</td>
</tr>
<tr>
<td>1</td>
<td>1 1.0000000</td>
</tr>
<tr>
<td>2</td>
<td>2 0.8888889</td>
</tr>
<tr>
<td>3</td>
<td>3 0.7777778</td>
</tr>
<tr>
<td>8</td>
<td>8 0.2222222</td>
</tr>
<tr>
<td>9</td>
<td>9 0.1111111</td>
</tr>
<tr>
<td>10</td>
<td>10 0.0000000</td>
</tr>
</tbody>
</table>

> mcols(gr)$score

[1] 1 2 3 4 5 6 7 8 9 10

Finally, the total lengths of the various sequences that the ranges are aligned to can also be stored in the GRanges object. So if this is data from Homo sapiens, we can set the values as:

> seqlengths(gr) <- c(249250621,243199373,198022430)

And then retrieves as:

> seqlengths(gr)

   chr1   chr2   chr3
249250621 243199373 198022430

Methods for accessing the length and names have also been defined.

> names(gr)

[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"

> length(gr)

[1] 10
2.1 Splitting and combining **GRanges** objects

GRanges objects can be divided into groups using the **split** method. This produces a **GRangesList** object, a class that will be discussed in detail in the next section.

```r
> sp <- split(gr, rep(1:2, each=5))
> sp
```

GRangesList object of length 2:

$1

GRanges object with 5 ranges and 2 metadata columns:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
<th>score</th>
<th>GC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&lt;Rle&gt;</td>
<td>&lt;IRanges&gt;</td>
<td>&lt;Rle&gt;</td>
<td>&lt;integer&gt;</td>
</tr>
<tr>
<td>a</td>
<td>chr1</td>
<td>[1, 7]</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>chr2</td>
<td>[2, 8]</td>
<td>+</td>
<td>2 0.8888888888888889</td>
</tr>
<tr>
<td>c</td>
<td>chr2</td>
<td>[3, 9]</td>
<td>+</td>
<td>3 0.7777777777777778</td>
</tr>
<tr>
<td>d</td>
<td>chr2</td>
<td>[4, 10]</td>
<td>*</td>
<td>4 0.6666666666666667</td>
</tr>
<tr>
<td>e</td>
<td>chr1</td>
<td>[5, 11]</td>
<td>*</td>
<td>5 0.5555555555555556</td>
</tr>
</tbody>
</table>

$2

GRanges object with 5 ranges and 2 metadata columns:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
<th>score</th>
<th>GC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&lt;Rle&gt;</td>
<td>&lt;IRanges&gt;</td>
<td>&lt;Rle&gt;</td>
<td>&lt;integer&gt;</td>
</tr>
<tr>
<td>f</td>
<td>chr1</td>
<td>[6, 12]</td>
<td>+</td>
<td>6 0.4444444444444444</td>
</tr>
<tr>
<td>g</td>
<td>chr3</td>
<td>[7, 13]</td>
<td>+</td>
<td>7 0.3333333333333333</td>
</tr>
<tr>
<td>h</td>
<td>chr3</td>
<td>[8, 14]</td>
<td>+</td>
<td>8 0.2222222222222222</td>
</tr>
<tr>
<td>i</td>
<td>chr3</td>
<td>[9, 15]</td>
<td>-</td>
<td>9 0.1111111111111111</td>
</tr>
<tr>
<td>j</td>
<td>chr3</td>
<td>[10, 16]</td>
<td>-</td>
<td>10 0</td>
</tr>
</tbody>
</table>

seqinfo: 3 sequences from an unspecified genome

If you then grab the components of this list, they can also be merged by using the `c` and `append` methods.

```r
> c(sp[[1]], sp[[2]])
```

GRanges object with 10 ranges and 2 metadata columns:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
<th>score</th>
<th>GC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&lt;Rle&gt;</td>
<td>&lt;IRanges&gt;</td>
<td>&lt;Rle&gt;</td>
<td>&lt;integer&gt;</td>
</tr>
<tr>
<td>a</td>
<td>chr1</td>
<td>[1, 7]</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>chr2</td>
<td>[2, 8]</td>
<td>+</td>
<td>2 0.8888888888888889</td>
</tr>
<tr>
<td>c</td>
<td>chr2</td>
<td>[3, 9]</td>
<td>+</td>
<td>3 0.7777777777777778</td>
</tr>
<tr>
<td>h</td>
<td>chr3</td>
<td>[8, 14]</td>
<td>+</td>
<td>8 0.2222222222222222</td>
</tr>
<tr>
<td>i</td>
<td>chr3</td>
<td>[9, 15]</td>
<td>-</td>
<td>9 0.1111111111111111</td>
</tr>
<tr>
<td>j</td>
<td>chr3</td>
<td>[10, 16]</td>
<td>-</td>
<td>10 0</td>
</tr>
</tbody>
</table>

seqinfo: 3 sequences from an unspecified genome

2.2 Subsetting **GRanges** objects

The expected subsetting operations are also available for **GRanges** objects.

```r
> gr[2:3]
```
GRanges object with 2 ranges and 2 metadata columns:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
<th>score</th>
<th>GC</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>chr2</td>
<td>[2, 8]</td>
<td>+</td>
<td>2 0.888888888888889</td>
</tr>
<tr>
<td>c</td>
<td>chr2</td>
<td>[3, 9]</td>
<td>+</td>
<td>3 0.777777777777778</td>
</tr>
</tbody>
</table>

seqinfo: 3 sequences from an unspecified genome

A second argument to the [ subset operator can be used to specify which metadata columns to extract from the GRanges object. For example,

```r
> gr[2:3, "GC"]
```

GRanges object with 2 ranges and 1 metadata column:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
<th>score</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>chr2</td>
<td>[2, 8]</td>
<td>+ 0.888888888888889</td>
</tr>
<tr>
<td>c</td>
<td>chr2</td>
<td>[3, 9]</td>
<td>+ 0.777777777777778</td>
</tr>
</tbody>
</table>

seqinfo: 3 sequences from an unspecified genome

You can also assign into elements of the GRanges object. Here is an example where the 2nd row of a GRanges object is replaced with the 1st row of gr.

```r
> singles <- split(gr, names(gr))
> grMod <- gr
> grMod[2] <- singles[[1]]
> head(grMod, n=3)
```

GRanges object with 3 ranges and 2 metadata columns:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
<th>score</th>
<th>GC</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>chr1</td>
<td>[1, 7]</td>
<td>-</td>
<td>1 1</td>
</tr>
<tr>
<td>b</td>
<td>chr1</td>
<td>[1, 7]</td>
<td>-</td>
<td>1 1</td>
</tr>
<tr>
<td>c</td>
<td>chr2</td>
<td>[3, 9]</td>
<td>+ 3 0.777777777777778</td>
<td></td>
</tr>
</tbody>
</table>

seqinfo: 3 sequences from an unspecified genome

Here is a second example where the metadata for score from the 3rd element is replaced with the score from the 2nd row etc.

```r
> grMod[2,1] <- singles[[3]][,1]
> head(grMod, n=3)
```

GRanges object with 3 ranges and 2 metadata columns:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
<th>score</th>
<th>GC</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>chr1</td>
<td>[1, 7]</td>
<td>-</td>
<td>1 1</td>
</tr>
<tr>
<td>b</td>
<td>chr2</td>
<td>[3, 9]</td>
<td>+ 3 0.777777777777778</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>chr2</td>
<td>[3, 9]</td>
<td>+ 3 0.777777777777778</td>
<td></td>
</tr>
</tbody>
</table>

seqinfo: 3 sequences from an unspecified genome

There are also methods to repeat, reverse, or select specific portions of GRanges objects.
> `rep(singles[[2]], times = 3)`

GRanges object with 3 ranges and 2 metadata columns:

```
  seqnames ranges strand | score   GC
  <Rle> <IRanges> <Rle> | <integer> <numeric>
  b   chr2 [2, 8] + | 2 0.888888888888889
  b   chr2 [2, 8] + | 2 0.888888888888889
  b   chr2 [2, 8] + | 2 0.888888888888889
```

seqinfo: 3 sequences from an unspecified genome

> `rev(gr)`

GRanges object with 10 ranges and 2 metadata columns:

```
  seqnames ranges strand | score   GC
  <Rle> <IRanges> <Rle> | <integer> <numeric>
  j   chr3 [10, 16] - | 10 0
  i   chr3 [ 9, 15] - | 9 0.111111111111111
  h   chr3 [ 8, 14] + | 8 0.222222222222222
```

> `head(gr,n=2)`

GRanges object with 2 ranges and 2 metadata columns:

```
  seqnames ranges strand | score   GC
  <Rle> <IRanges> <Rle> | <integer> <numeric>
  a   chr1 [1, 7] - | 1 1
  b   chr2 [2, 8] + | 2 0.888888888888889
```

seqinfo: 3 sequences from an unspecified genome

> `tail(gr,n=2)`

GRanges object with 2 ranges and 2 metadata columns:

```
  seqnames ranges strand | score   GC
  <Rle> <IRanges> <Rle> | <integer> <numeric>
  i   chr3 [ 9, 15] - | 9 0.111111111111111
  j   chr3 [10, 16] - | 10 0
```

seqinfo: 3 sequences from an unspecified genome

> `window(gr, start=2,end=4)`

GRanges object with 3 ranges and 2 metadata columns:

```
  seqnames ranges strand | score   GC
  <Rle> <IRanges> <Rle> | <integer> <numeric>
  b   chr2 [2, 8] + | 2 0.888888888888889
  c   chr2 [3, 9] + | 3 0.777777777777778
```
2.3 Basic interval operations for \textit{GRanges} objects

Basic interval characteristics of \textit{GRanges} objects can be extracted using the \texttt{start}, \texttt{end}, \texttt{width}, and \texttt{range} methods.

\begin{verbatim}
> g <- gr[1:3]
> g <- append(g, singles[[10]])
> start(g)
[1] 1 2 3 10
> end(g)
[1] 7 8 9 16
> width(g)
[1] 7 7 7 7
> range(g)
\end{verbatim}

The \textit{GRanges} class also has many methods for manipulating the intervals. For example, the \texttt{flank} method can be used to recover regions flanking the set of ranges represented by the \textit{GRanges} object. So to get a \textit{GRanges} object containing the ranges that include the 10 bases upstream of the ranges:

\begin{verbatim}
> flank(g, 10)
\end{verbatim}
GRanges object with 4 ranges and 2 metadata columns:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
<th>score</th>
<th>GC</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>chr1</td>
<td>[-8, 17]</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>chr2</td>
<td>[-8, 1]</td>
<td>+</td>
<td>2</td>
</tr>
<tr>
<td>c</td>
<td>chr2</td>
<td>[-7, 2]</td>
<td>+</td>
<td>3</td>
</tr>
<tr>
<td>j</td>
<td>chr3</td>
<td>[17, 26]</td>
<td>-</td>
<td>10</td>
</tr>
</tbody>
</table>

---

seqinfo: 3 sequences from an unspecified genome

And to include the downstream bases:

```r
> flank(g, 10, start=FALSE)
```

GRanges object with 4 ranges and 2 metadata columns:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
<th>score</th>
<th>GC</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>chr1</td>
<td>[-9, 0]</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>chr2</td>
<td>[9, 18]</td>
<td>+</td>
<td>2</td>
</tr>
<tr>
<td>c</td>
<td>chr2</td>
<td>[10, 19]</td>
<td>+</td>
<td>3</td>
</tr>
<tr>
<td>j</td>
<td>chr3</td>
<td>[0, 9]</td>
<td>-</td>
<td>10</td>
</tr>
</tbody>
</table>

---

seqinfo: 3 sequences from an unspecified genome

Similar to `flank`, there are also operations to `resize` and `shift` our `GRanges` object. The `shift` method will move the ranges by a specific number of base pairs, and the `resize` method will extend the ranges by a specified width.

```r
> shift(g, 5)
```

GRanges object with 4 ranges and 2 metadata columns:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
<th>score</th>
<th>GC</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>chr1</td>
<td>[6, 12]</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>chr2</td>
<td>[7, 13]</td>
<td>+</td>
<td>2</td>
</tr>
<tr>
<td>c</td>
<td>chr2</td>
<td>[8, 14]</td>
<td>+</td>
<td>3</td>
</tr>
<tr>
<td>j</td>
<td>chr3</td>
<td>[15, 21]</td>
<td>-</td>
<td>10</td>
</tr>
</tbody>
</table>

---

seqinfo: 3 sequences from an unspecified genome

```r
> resize(g, 30)
```

GRanges object with 4 ranges and 2 metadata columns:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
<th>score</th>
<th>GC</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>chr1</td>
<td>[-22, 7]</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>chr2</td>
<td>[2, 31]</td>
<td>+</td>
<td>2</td>
</tr>
<tr>
<td>c</td>
<td>chr2</td>
<td>[3, 32]</td>
<td>+</td>
<td>3</td>
</tr>
<tr>
<td>j</td>
<td>chr3</td>
<td>[-13, 16]</td>
<td>-</td>
<td>10</td>
</tr>
</tbody>
</table>

---

seqinfo: 3 sequences from an unspecified genome

The `reduce` will align the ranges and merge overlapping ranges to produce a simplified set.
> reduce(g)

GRanges object with 3 ranges and 0 metadata columns:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&lt;IRanges&gt;</td>
<td></td>
</tr>
<tr>
<td>[1]</td>
<td>chr1 [1, 7]</td>
<td>-</td>
</tr>
<tr>
<td>[3]</td>
<td>chr3 [10, 16]</td>
<td>-</td>
</tr>
</tbody>
</table>

seqinfo: 3 sequences from an unspecified genome

Sometimes you may be interested in the spaces or the qualities of the spaces between the ranges represented by your GRanges object. The gaps method will help you calculate the spaces between a reduced version of your ranges:

> gaps(g)

GRanges object with 11 ranges and 0 metadata columns:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&lt;Rle&gt; &lt;IRanges&gt;</td>
<td>&lt;Rle&gt;</td>
</tr>
<tr>
<td>[1]</td>
<td>chr1 [1, 249250621]</td>
<td>+</td>
</tr>
<tr>
<td>[2]</td>
<td>chr1 [8, 249250621]</td>
<td>-</td>
</tr>
<tr>
<td>[3]</td>
<td>chr1 [1, 249250621]</td>
<td>*</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>[9]</td>
<td>chr3 [1, 9]</td>
<td>-</td>
</tr>
<tr>
<td>[10]</td>
<td>chr3 [17, 198022430]</td>
<td>-</td>
</tr>
</tbody>
</table>

seqinfo: 3 sequences from an unspecified genome

And sometimes you also may want to know how many quantitatively unique fragments your ranges could possibly represent. For this task there is the disjoin method.

> disjoin(g)

GRanges object with 5 ranges and 0 metadata columns:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&lt;IRanges&gt;</td>
<td></td>
</tr>
<tr>
<td>[1]</td>
<td>chr1 [1, 7]</td>
<td>-</td>
</tr>
<tr>
<td>[2]</td>
<td>chr2 [2, 2]</td>
<td>+</td>
</tr>
<tr>
<td>[3]</td>
<td>chr2 [3, 8]</td>
<td>+</td>
</tr>
<tr>
<td>[4]</td>
<td>chr2 [9, 9]</td>
<td>+</td>
</tr>
<tr>
<td>[5]</td>
<td>chr3 [10, 16]</td>
<td>-</td>
</tr>
</tbody>
</table>

seqinfo: 3 sequences from an unspecified genome

One of the most powerful methods for looking at GRanges objects is the coverage method. The coverage method quantifies the degree of overlap for all the ranges in a GRanges object.

> coverage(g)

RleList of length 3
$chr1
2.4 Interval set operations for \textit{GRanges} objects

There are also operations for calculating relationships between different \textit{GRanges} objects. Here are some examples for how you can calculate the union, the intersect and the asymmetric difference (using \texttt{setdiff}).

```r
> g2 <- head(gr, n=2)
> union(g, g2)
```

\texttt{GRanges} object with 3 ranges and 0 metadata columns:

\begin{verbatim}
   seqnames ranges strand
   <Rle> <IRanges> <Rle>
[1] chr1 [ 1, 7] -
[3] chr3 [10, 16] -
```

\texttt{seqinfo: 3 sequences from an unspecified genome}

```r
> intersect(g, g2)
```

\texttt{GRanges} object with 2 ranges and 0 metadata columns:

\begin{verbatim}
   seqnames ranges strand
   <Rle> <IRanges> <Rle>
[1] chr1 [ 1, 7] -
[2] chr2 [ 2, 8] +
```

\texttt{seqinfo: 3 sequences from an unspecified genome}

```r
> setdiff(g, g2)
```

\texttt{GRanges} object with 2 ranges and 0 metadata columns:

\begin{verbatim}
   seqnames ranges strand
   <Rle> <IRanges> <Rle>
[1] chr2 [ 9, 9] +
[2] chr3 [10, 16] -
```

\texttt{seqinfo: 3 sequences from an unspecified genome}

In addition, there is similar set of operations that act at the level of the individual ranges within each \textit{GRanges}. These operations all begin with a “p”, which is short for parallel. A requirement for this set of operations is that the number of elements in each \textit{GRanges} object has to be the same, and that both of the objects have to have the same seqnames and strand assignments throughout.

> g3 <- g[1:2]
> ranges(g3[1]) <- IRanges(start=5, end=12)
> punion(g2, g3)

GRanges object with 2 ranges and 0 metadata columns:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>chr1 [1, 12]</td>
<td>-</td>
</tr>
<tr>
<td>b</td>
<td>chr2 [2, 8]</td>
<td>+</td>
</tr>
</tbody>
</table>

seqinfo: 3 sequences from an unspecified genome

> pintersect(g2, g3)

GRanges object with 2 ranges and 3 metadata columns:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
<th>score</th>
<th>GC</th>
<th>hit</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>chr1 [5, 7]</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>chr2 [2, 8]</td>
<td>+</td>
<td>2</td>
<td>0.888888888888889</td>
<td>1</td>
</tr>
</tbody>
</table>

seqinfo: 3 sequences from an unspecified genome

> psetdiff(g2, g3)

GRanges object with 2 ranges and 0 metadata columns:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>chr1 [1, 4]</td>
<td>-</td>
</tr>
<tr>
<td>b</td>
<td>chr2 [2, 1]</td>
<td>+</td>
</tr>
</tbody>
</table>

seqinfo: 3 sequences from an unspecified genome

For even more information on the GRanges classes be sure to consult the manual page.

> ?GRanges

3 **GRangesList**: Groups of Genomic Ranges

Some important genomic features, such as spliced transcripts that are comprised of exons, are inherently compound structures. Such a feature makes much more sense when expressed as a compound object such as a **GRangesList**. Whenever genomic features consist of multiple ranges that are grouped by a parent feature, they can be represented as a **GRangesList** object. Consider the simple example of the two transcript **GRangesList** below created using the **GRangesList** constructor.

```r
> gr1 <-
+ GRanges(seqnames = "chr2", ranges = IRanges(3, 6),
+         strand = "+", score = 5L, GC = 0.45)
> gr2 <-
+ GRanges(seqnames = c("chr1", "chr1"),
+         ranges = IRanges(c(7,13), width = 3),
+         strand = c("+", "-"), score = 3:4, GC = c(0.3, 0.5))
> grl <- GRangesList("txA" = gr1, "txB" = gr2)
> grl
```
GRangesList object of length 2:

$txA
GRanges object with 1 range and 2 metadata columns:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
<th>score</th>
<th>GC</th>
</tr>
</thead>
<tbody>
<tr>
<td>chr2</td>
<td>[3, 6]</td>
<td>+</td>
<td>5</td>
<td>0.45</td>
</tr>
</tbody>
</table>

$txB
GRanges object with 2 ranges and 2 metadata columns:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
<th>score</th>
<th>GC</th>
</tr>
</thead>
<tbody>
<tr>
<td>chr1</td>
<td>[7, 9]</td>
<td>+</td>
<td>3</td>
<td>0.3</td>
</tr>
<tr>
<td>chr1</td>
<td>[13, 15]</td>
<td>-</td>
<td>4</td>
<td>0.5</td>
</tr>
</tbody>
</table>

seqinfo: 2 sequences from an unspecified genome; no seqlengths

The `show` method for a `GRangesList` object displays it as a named list of `GRanges` objects, where the names of this list are considered to be the names of the grouping feature. In the example above, the groups of individual exon ranges are represented as separate `GRanges` objects which are further organized into a list structure where each element name is a transcript name. Many other combinations of grouped and labeled `GRanges` objects are possible of course, but this example is expected to be a common arrangement.

3.1 Basic `GRangesList` accessors

Just as with `GRanges` object, the components of the genomic coordinates within a `GRangesList` object can be extracted using simple accessor methods. Not surprisingly, the `GRangesList` objects have many of the same accessors as `GRanges` objects. The difference is that many of these methods return a list since the input is now essentially a list of `GRanges` objects. Here are a few examples:

```r
> seqnames(grl)
RleList of length 2
$txA
factor-Rle of length 1 with 1 run
  Lengths: 1
  Values : chr2
  Levels(2): chr2 chr1

$txB
factor-Rle of length 2 with 1 run
  Lengths: 2
  Values : chr1
  Levels(2): chr2 chr1
```

```r
> ranges(grl)
IRangesList of length 2
$txA
IRanges object with 1 range and 0 metadata columns:

<table>
<thead>
<tr>
<th>start</th>
<th>end</th>
<th>width</th>
</tr>
</thead>
<tbody>
<tr>
<td>[3]</td>
<td>6</td>
<td>4</td>
</tr>
</tbody>
</table>
```
$txB
IRanges object with 2 ranges and 0 metadata columns:

<table>
<thead>
<tr>
<th>start</th>
<th>end</th>
<th>width</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>13</td>
<td>15</td>
<td>3</td>
</tr>
</tbody>
</table>

> strand(grl)
RleList of length 2

$txA
factor-Rle of length 1 with 1 run
- Lengths: 1
- Values : +
- Levels(3): + - *

$txB
factor-Rle of length 2 with 2 runs
- Lengths: 1 1
- Values : + -
- Levels(3): + - *

The \texttt{length} and \texttt{names} methods will return the length or names of the list and the \texttt{seqlengths} method will return the set of sequence lengths.

> length(grl)
[1] 2

> names(grl)
[1] "txA" "txB"

> seqlengths(grl)
chr2 chr1
NA NA

The \texttt{elementNROWS} method returns a list of integers corresponding to the result of calling \texttt{NROW} on each individual \texttt{GRanges} object contained by the \texttt{GRangesList}. This is a faster alternative to calling \texttt{lapply} on the \texttt{GRangesList}.

> elementNROWS(grl)
txA txB
1 2

You can also use \texttt{isEmpty} to test if a \texttt{GRangesList} object contains anything.

> isEmpty(grl)
[1] FALSE
Finally, in the context of a `GRangesList` object, the `mcols` method performs a similar operation to what it does on a `GRanges` object. However, this metadata now refers to information at the list level instead of the level of the individual `GRanges` objects.

```r
> mcols(grl) <- c("Transcript A","Transcript B")
> mcols(grl)
```

Dataframe with 2 rows and 1 column

<table>
<thead>
<tr>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transcript A</td>
</tr>
<tr>
<td>Transcript B</td>
</tr>
</tbody>
</table>

### 3.2 Combining `GRangesList` objects

`GRangesList` objects can be unlisted to combine the separate `GRanges` objects that they contain as an expanded `GRanges`.

```r
> ul <- unlist(grl)
```

You can also append values together using `append` or `c`.

### 3.3 Basic interval operations for `GRangesList` objects

For interval operations, many of the same methods exist for `GRangesList` objects that exist for `GRanges` objects.

```r
> start(grl)
```

IntegerList of length 2

```
[["txA"]]
3

[["txB"]]
7 13
```

```r
> end(grl)
```

IntegerList of length 2

```
[["txA"]]
6

[["txB"]]
9 15
```

```r
> width(grl)
```

IntegerList of length 2

```
[["txA"]]
4

[["txB"]]
3 3
```

And as with `GRanges` objects, you can also shift all the `GRanges` objects in a `GRangesList` object, or calculate the coverage. Both of these operations are also carried out across each `GRanges` list member.

```r
> shift(grl, 20)
```

GRangesList object of length 2:

```
$txA
GRanges object with 1 range and 2 metadata columns:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges strand</th>
<th>score</th>
<th>GC</th>
</tr>
</thead>
</table>
```
$txB
GRanges object with 2 ranges and 2 metadata columns:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
<th>score</th>
<th>GC</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>chr1</td>
<td>[27, 29]</td>
<td>+</td>
<td>3 0.3</td>
</tr>
<tr>
<td>[2]</td>
<td>chr1</td>
<td>[33, 35]</td>
<td>-</td>
<td>4 0.5</td>
</tr>
</tbody>
</table>

------
seqinfo: 2 sequences from an unspecified genome; no seqlengths

> coverage(grl)

RleList of length 2
$chr2
integer-Rle of length 6 with 2 runs
Lengths: 2 4
Values: 0 1

$chr1
integer-Rle of length 15 with 4 runs
Lengths: 6 3 3 3
Values: 0 1 0 1

### 3.4 Subsetting `GRangesList` objects

As you might guess, the subsetting of a `GRangesList` object is quite different from subsetting on a `GRanges` object in that it acts as if you are subseting a list. If you try out the following you will notice that the standard conventions have been followed.

> grl[1]
> grl[[1]]
> grl["txA"]
> grl$txB

But in addition to this, when subsetting a `GRangesList`, you can also pass in a second parameter (as with a `GRanges` object) to again specify which of the metadata columns you wish to select.

> grl[1, "score"]

GRangesList object of length 1:
$txA
GRanges object with 1 range and 1 metadata column:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
<th>score</th>
</tr>
</thead>
</table>

------
seqinfo: 2 sequences from an unspecified genome; no seqlengths

> grl["txB", "GC"]

15
GRangesList object of length 1:
	$txB
GRanges object with 2 ranges and 1 metadata column:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
<th>GC</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Rle&gt;</td>
<td>&lt;IRanges&gt;</td>
<td>&lt;Rle&gt;</td>
<td>&lt;numeric&gt;</td>
</tr>
<tr>
<td>[1] chr1</td>
<td>[7, 9]</td>
<td>+</td>
<td>0.3</td>
</tr>
</tbody>
</table>

seqinfo: 2 sequences from an unspecified genome; no seqlengths

The `head`, `tail`, `rep`, `rev`, and `window` methods all behave as you would expect them to for a list object. For example, the elements referred to by `window` are now list elements instead of `GRanges` elements.

> `rep(grl[[1]], times = 3)`

GRanges object with 3 ranges and 2 metadata columns:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
<th>score</th>
<th>GC</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Rle&gt;</td>
<td>&lt;IRanges&gt;</td>
<td>&lt;Rle&gt;</td>
<td>&lt;integer&gt;</td>
<td>&lt;numeric&gt;</td>
</tr>
<tr>
<td>[1] chr2</td>
<td>[3, 6]</td>
<td>+</td>
<td>5</td>
<td>0.45</td>
</tr>
<tr>
<td>[2] chr2</td>
<td>[3, 6]</td>
<td>+</td>
<td>5</td>
<td>0.45</td>
</tr>
<tr>
<td>[3] chr2</td>
<td>[3, 6]</td>
<td>+</td>
<td>5</td>
<td>0.45</td>
</tr>
</tbody>
</table>

seqinfo: 2 sequences from an unspecified genome; no seqlengths

> `rev(grl)`

GRangesList object of length 2:
	$txB
GRanges object with 2 ranges and 2 metadata columns:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
<th>score</th>
<th>GC</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Rle&gt;</td>
<td>&lt;IRanges&gt;</td>
<td>&lt;Rle&gt;</td>
<td>&lt;integer&gt;</td>
<td>&lt;numeric&gt;</td>
</tr>
<tr>
<td>[1] chr1</td>
<td>[7, 9]</td>
<td>+</td>
<td>3</td>
<td>0.3</td>
</tr>
</tbody>
</table>

$txA
GRanges object with 1 range and 2 metadata columns:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
<th>score</th>
<th>GC</th>
</tr>
</thead>
<tbody>
<tr>
<td>chr2</td>
<td>[3, 6]</td>
<td>+</td>
<td>5</td>
<td>0.45</td>
</tr>
</tbody>
</table>

seqinfo: 2 sequences from an unspecified genome; no seqlengths

> `head(grl, n=1)`

GRangesList object of length 1:
	$txA
GRanges object with 1 range and 2 metadata columns:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
<th>score</th>
<th>GC</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Rle&gt;</td>
<td>&lt;IRanges&gt;</td>
<td>&lt;Rle&gt;</td>
<td>&lt;integer&gt;</td>
<td>&lt;numeric&gt;</td>
</tr>
<tr>
<td>[1] chr2</td>
<td>[3, 6]</td>
<td>+</td>
<td>5</td>
<td>0.45</td>
</tr>
</tbody>
</table>

seqinfo: 2 sequences from an unspecified genome; no seqlengths
> tail(gr1, n=1)

GRangesList object of length 1:

$txB

GRanges object with 2 ranges and 2 metadata columns:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
<th>score</th>
<th>GC</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Rle&gt;</td>
<td>&lt;IRanges&gt;</td>
<td>&lt;Rle&gt;</td>
<td>&lt;integer&gt;</td>
<td>&lt;numeric&gt;</td>
</tr>
<tr>
<td>[1] chr1</td>
<td>[ 7, 9]</td>
<td>+</td>
<td>3</td>
<td>0.3</td>
</tr>
</tbody>
</table>

seqinfo: 2 sequences from an unspecified genome; no seqlengths

> window(gr1, start=1, end=1)

GRangesList object of length 1:

$txA

GRanges object with 1 range and 2 metadata columns:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
<th>score</th>
<th>GC</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Rle&gt;</td>
<td>&lt;IRanges&gt;</td>
<td>&lt;Rle&gt;</td>
<td>&lt;integer&gt;</td>
<td>&lt;numeric&gt;</td>
</tr>
<tr>
<td>[1] chr2</td>
<td>[3, 6]</td>
<td>+</td>
<td>5</td>
<td>0.45</td>
</tr>
</tbody>
</table>

seqinfo: 2 sequences from an unspecified genome; no seqlengths

> gr1[IRanges(start=2, end=2)]

GRangesList object of length 1:

$txB

GRanges object with 2 ranges and 2 metadata columns:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
<th>score</th>
<th>GC</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Rle&gt;</td>
<td>&lt;IRanges&gt;</td>
<td>&lt;Rle&gt;</td>
<td>&lt;integer&gt;</td>
<td>&lt;numeric&gt;</td>
</tr>
<tr>
<td>[1] chr1</td>
<td>[ 7, 9]</td>
<td>+</td>
<td>3</td>
<td>0.3</td>
</tr>
</tbody>
</table>

seqinfo: 2 sequences from an unspecified genome; no seqlengths

3.5 Looping over *GRangesList* objects

For *GRangesList* objects there is also a family of apply methods. These include lapply, sapply, mapply, endoapply, mendoapply, Map, and Reduce.

The different looping methods defined for *GRangesList* objects are useful for returning different kinds of results. The standard lapply and sapply behave according to convention, with the lapply method returning a list and sapply returning a more simplified output.

> lapply(gr1, length)

$txA

[1] 1

$txB

[1] 2
As with `IRanges` objects, there is also a multivariate version of `sapply`, called `mapply`, defined for `GRangesList` objects. And, if you don’t want the results simplified, you can call the `Map` method, which does the same things as `mapply` but without simplifying the output.

```r
> grl2 <- shift(grl, 10)
> names(grl2) <- c("shiftTxA", "shiftTxB")
> mapply(c, grl, grl2)
```

```
$txA
GRanges object with 2 ranges and 2 metadata columns:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
<th>score</th>
<th>GC</th>
</tr>
</thead>
</table>
| chr2     | [ 3, 6]      | +      | 5     | 0.45
| chr2     | [13, 16]     | +      | 5     | 0.45

 seqinfo: 2 sequences from an unspecified genome; no seqlengths

$txB
GRanges object with 4 ranges and 2 metadata columns:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
<th>score</th>
<th>GC</th>
</tr>
</thead>
<tbody>
<tr>
<td>chr1</td>
<td>[ 7, 9]</td>
<td>+</td>
<td>3</td>
<td>0.3</td>
</tr>
<tr>
<td>chr1</td>
<td>[13, 15]</td>
<td>-</td>
<td>4</td>
<td>0.5</td>
</tr>
<tr>
<td>chr1</td>
<td>[17, 19]</td>
<td>+</td>
<td>3</td>
<td>0.3</td>
</tr>
<tr>
<td>chr1</td>
<td>[23, 25]</td>
<td>-</td>
<td>4</td>
<td>0.5</td>
</tr>
</tbody>
</table>

 seqinfo: 2 sequences from an unspecified genome; no seqlengths

> Map(c, grl, grl2)
```

```
$txA
GRanges object with 2 ranges and 2 metadata columns:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
<th>score</th>
<th>GC</th>
</tr>
</thead>
</table>
| chr2     | [ 3, 6]      | +      | 5     | 0.45
| chr2     | [13, 16]     | +      | 5     | 0.45

 seqinfo: 2 sequences from an unspecified genome; no seqlengths

$txB
GRanges object with 4 ranges and 2 metadata columns:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
<th>score</th>
<th>GC</th>
</tr>
</thead>
<tbody>
<tr>
<td>chr1</td>
<td>[ 7, 9]</td>
<td>+</td>
<td>3</td>
<td>0.3</td>
</tr>
<tr>
<td>chr1</td>
<td>[13, 15]</td>
<td>-</td>
<td>4</td>
<td>0.5</td>
</tr>
<tr>
<td>chr1</td>
<td>[17, 19]</td>
<td>+</td>
<td>3</td>
<td>0.3</td>
</tr>
<tr>
<td>chr1</td>
<td>[23, 25]</td>
<td>-</td>
<td>4</td>
<td>0.5</td>
</tr>
</tbody>
</table>

 seqinfo: 2 sequences from an unspecified genome; no seqlengths
Sometimes, you may not want to get back a simplified output or a list. Sometimes you will want to get back a modified version of the \texttt{GRangesList} that you originally passed in. This is conceptually similar to the mathematical notion of an endomorphism. This is achieved using the \texttt{endoapply} method, which will return the results as a \texttt{GRangesList} object.

\begin{verbatim}
> endoapply(grl,rev)

GRangesList object of length 2:
$txA
GRanges object with 1 range and 2 metadata columns:
\begin{verbatim}
seqnames ranges strand | score GC
<Rle> <IRanges> <Rle> | <integer> <numeric>
[1] chr2 [3, 6] + | 5 0.45
\end{verbatim}

$txB
GRanges object with 2 ranges and 2 metadata columns:
\begin{verbatim}
seqnames ranges strand | score GC
[1] chr1 [13, 15] - | 4 0.5
[2] chr1 [ 7, 9] + | 3 0.3
\end{verbatim}

------
seqinfo: 2 sequences from an unspecified genome; no seqlengths

And, there is also a multivariate version of the \texttt{endoapply} method in the form of the \texttt{mendoapply} method.

\begin{verbatim}
> mendoapply(c,grl,grl2)

GRangesList object of length 2:
$txA
GRanges object with 2 ranges and 2 metadata columns:
\begin{verbatim}
seqnames ranges strand | score GC
<Rle> <IRanges> <Rle> | <integer> <numeric>
[1] chr2 [3, 6] + | 5 0.45
[2] chr2 [13, 16] + | 5 0.45
\end{verbatim}

$txB
GRanges object with 4 ranges and 2 metadata columns:
\begin{verbatim}
seqnames ranges strand | score GC
[1] chr1 [ 7, 9] + | 3 0.3
[3] chr1 [17, 19] + | 3 0.3
\end{verbatim}

------
seqinfo: 2 sequences from an unspecified genome; no seqlengths

Finally, the \texttt{Reduce} method will allow the \texttt{GRanges} objects to be collapsed across the whole of the \texttt{GRangesList} object.

\begin{verbatim}
> Reduce(c,grl)

GRanges object with 3 ranges and 2 metadata columns:
\begin{verbatim}
seqnames ranges strand | score GC
19
\end{verbatim}
\end{verbatim}
seqinfo: 2 sequences from an unspecified genome; no seqlengths

For even more information on the GRangesList classes be sure to consult the manual page.

> ?GRangesList

**4 Interval overlaps involving GRanges and GRangesList objects**

Interval overlapping is the process of comparing the ranges in two objects to determine if and when they overlap. As such, it is perhaps the most common operation performed on GRanges and GRangesList objects. To this end, the GenomicRanges package provides a family of interval overlap functions. The most general of these functions is findOverlaps, which takes a query and a subject as inputs and returns a Hits object containing the index pairings for the overlapping elements.

> mtc <- findOverlaps(gr, grl)
> as.matrix(mtc)

```
queryHits subjectHits
[1,] 2 1
[2,] 3 1
[3,] 4 1
[4,] 5 2
[5,] 6 2
```

As suggested in the sections discussing the nature of the GRanges and GRangesList classes, the index in the above matrix of hits for a GRanges object is a single range while for a GRangesList object it is the set of ranges that define a “feature”.

Another function in the overlaps family is countOverlaps, which tabulates the number of overlaps for each element in the query.

> countOverlaps(gr, grl)

```
a b c d e f g h i j
0 1 1 1 1 0 0 0 0 0
```

A third function in this family is subsetByOverlaps, which extracts the elements in the query that overlap at least one element in the subject.

> subsetByOverlaps(gr, grl)

**GRanges object with 5 ranges and 2 metadata columns:**

```
<IRanges> <IRanges> <numeric>
[1] chr2 [ 3, 6] + | 5 0.45
[2] chr1 [ 7, 9] + | 3 0.3
-------
seqinfo: 3 sequences from an unspecified genome
```
Finally, you can use the `select` argument to get the index of the first overlapping element in the subject for each element in the query.

```r
> findOverlaps(gr, gr1, select="first")
[1] NA 1 1 1 2 2 NA NA NA NA
> findOverlaps(gr1, gr, select="first")
[1] 2 5
```

## 5 Session Information

All of the output in this vignette was produced under the following conditions:

```r
> sessionInfo()
```

```
R version 3.3.1 (2016-06-21)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 16.04.1 LTS

locale:
[1] LC_CTYPE=en_US.UTF-8     LC_NUMERIC=C
[3] LC_TIME=en_US.UTF-8      LC_COLLATE=C
[5] LC_MONETARY=en_US.UTF-8  LC_MESSAGES=en_US.UTF-8
[7] LC_PAPER=en_US.UTF-8     LC_NAME=C
[9] LC_ADDRESS=C             LC_TELEPHONE=C

attached base packages:
[1] parallel  stats4   stats   graphics  grDevices  utils
[7] datasets  methods  base

other attached packages:
[1] BSgenome.Scerevisiae.UCSC.sacCer2_1.4.0
[2] KEGGgraph_1.32.0
[3] KEGG.db_3.2.3
[4] BSgenome.Hsapiens.UCSC hg19_1.4.0
[5] BSgenome_1.42.0
[6] rtracklayer_1.34.0
[7] edgeR_3.16.0
[8] limma_3.30.0
[9] DESeq2_1.14.0
[10] AnnotationHub_2.6.0
[12] TxDb.Hsapiens.UCSC.hg19.knownGene_3.2.2
[14] GenomicFeatures_1.26.0
[15] AnnotationDbi_1.36.0
[16] GenomicAlignments_1.10.0
[17] Rsamtools_1.26.0
[18] Biostrings_2.42.0
[19] XVector_0.14.0
[20] SummarizedExperiment_1.4.0
[21] Biobase_2.34.0
[22] pasillaBamSubset_0.11.0
```
<table>
<thead>
<tr>
<th>Package</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>GenomicRanges_1.26.1</td>
<td></td>
</tr>
<tr>
<td>GenomeInfoDb_1.10.0</td>
<td></td>
</tr>
<tr>
<td>IRanges_2.8.0</td>
<td></td>
</tr>
<tr>
<td>S4Vectors_0.12.0</td>
<td></td>
</tr>
<tr>
<td>BiocGenerics_0.20.0</td>
<td></td>
</tr>
<tr>
<td>locfit_1.5-9.1</td>
<td></td>
</tr>
<tr>
<td>lattice_0.20-34</td>
<td></td>
</tr>
<tr>
<td>mime_0.5</td>
<td></td>
</tr>
<tr>
<td>plyr_1.8.4</td>
<td></td>
</tr>
<tr>
<td>acepack_1.3-3.3</td>
<td></td>
</tr>
<tr>
<td>htrtr_1.2.1</td>
<td></td>
</tr>
<tr>
<td>BiocInstaller_1.24.0</td>
<td></td>
</tr>
<tr>
<td>curl_1.2.1</td>
<td></td>
</tr>
<tr>
<td>annotate_1.52.0</td>
<td></td>
</tr>
<tr>
<td>Matrix_1.2-7.1</td>
<td></td>
</tr>
<tr>
<td>BiocParallel_1.8.0</td>
<td></td>
</tr>
<tr>
<td>foreign_0.8-67</td>
<td></td>
</tr>
<tr>
<td>biomaRt_2.30.0</td>
<td></td>
</tr>
<tr>
<td>shiny_0.14.1</td>
<td></td>
</tr>
<tr>
<td>htmltools_0.3.5</td>
<td></td>
</tr>
<tr>
<td>gridExtra_2.2.1</td>
<td></td>
</tr>
<tr>
<td>Hmisc_3.17-4</td>
<td></td>
</tr>
<tr>
<td>bitops_1.0-6</td>
<td></td>
</tr>
<tr>
<td>xtable_1.8-2</td>
<td></td>
</tr>
<tr>
<td>DBI_0.5-1</td>
<td></td>
</tr>
<tr>
<td>graph_1.52.0</td>
<td></td>
</tr>
<tr>
<td>latticeExtra_0.6-28</td>
<td></td>
</tr>
<tr>
<td>BiocStyle_2.2.0</td>
<td></td>
</tr>
<tr>
<td>tools_3.3.1</td>
<td></td>
</tr>
<tr>
<td>colorspace_1.2-7</td>
<td></td>
</tr>
<tr>
<td>VariantAnnotation_1.20.0</td>
<td></td>
</tr>
</tbody>
</table>