An overview of the *girafe* package

J. Toedling, C. Ciaudo, O. Voinnet, E. Heard, E. Barillot

April 24, 2017

1 Introduction

The intent of package *girafe* is to facilitate the functional exploration of the alignments of multiple reads\(^1\) from next-generation sequencing (NGS) data to a genome.

It extends the functionality of the Bioconductor (Gentleman et al., 2004) packages Short-Read (Morgan et al., 2009) and *genomeIntervals*.

```r
> library("girafe")
> library("RColorBrewer")
```

If you use *girafe* for analysing your data, please cite:


Getting help

If possible, please send questions about *girafe* to the Bioconductor mailing list. See [http://www.bioconductor.org/docs/mailList.html](http://www.bioconductor.org/docs/mailList.html) Their archive of questions and responses may prove helpful, too.

2 Workflow

We present the functionality of the package *girafe* using example data that was downloaded from the Gene Expression Omnibus (GEO) database (Edgar et al., 2002, GSE10364). The example data are Solexa reads of 26 nt length derived from small RNA profiling of mouse oocytes. The data has previously been described in Tam et al. (2008).

```r
> exDir <- system.file("extdata", package="girafe")
> ### load object describing annotated mouse genome features:
> load(file.path(exDir, "mgi_gi.RData"))
```

\(^1\) The package has been developed for analysing single-end reads (fragment libraries) and does not support mate-pair reads yet.
2.1 Adapter trimming

We load reads that include parts of the adapter sequence.

```r
> ra23.wa <- readFastq(dirPath=exDir, pattern=
  + "aravinSRNA_23_plus_adapter_excerpt.fastq")
```

```r
> show(ra23.wa)
```

```
class: ShortReadQ
length: 1000 reads; width: 26 cycles
```

To removing adapter sequences, we use the function `trimAdapter`, which relies on the `pairwiseAlignment` function from the `Biostrings` package. The adapter sequence was obtained from the GEO page of the data.

```r
> adapter <- "CTGTAGGCCACCATCAAT"
> ra23.na <- trimAdapter(ra23.wa, adapter)
> show(ra23.na)
```

```
class: ShortReadQ
length: 1000 reads; width: 23 cycles
```

2.2 Importing aligned reads

The reads have been mapped to the mouse genome (assembly mm9) using the alignment tool `Bowtie` alignment tool (Langmead et al., 2009).

The resulting tab-delimited map file can be read into an object of class `AlignedRead` using the function `readAligned`. Both, this class and this function, are defined in the Bioconductor package `ShortRead`.

```r
> exA <- readAligned(dirPath=exDir, type="Bowtie",
  + pattern="aravinSRNA_23_no_adapter_excerpt_mm9_unmasked.bwtmap")
> show(exA)
```

```
class: AlignedRead
length: 1689 reads; width: 23 cycles
chromosome: chr14 chr17 ... chr3 chr14
position: 115443405 13011859 ... 68813840 62250772
strand: + + ... + -
alignQuality: NumericQuality
alignData varLabels: similar mismatch
```

The object of class `AlignedRead` can be converted into an object of class `AlignedGenomeIntervals`, which is the main class of the `giraff` package.
> exAI <- as(exA, "AlignedGenomeIntervals")
> organism(exAI) <- "Mm"

For alignments in BAM format (Li et al., 2009), there is an alternative way of importing the data. The function `agiFromBam` can be used to directly create `AlignedGenomeIntervals` objects from indexed and sorted BAM files, making use of functionalities in the `Rsamtools` package.

### 2.3 Exploring the aligned reads

> show(exAI)

1,286 genome intervals with 1,689 aligned reads on 22 chromosome(s).

Organism: Mm

Which chromosomes are the intervals located on?

> table(seqnames(exAI))

<table>
<thead>
<tr>
<th>seqName</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>chr1</td>
<td>112</td>
</tr>
<tr>
<td>chr10</td>
<td>50</td>
</tr>
<tr>
<td>chr11</td>
<td>60</td>
</tr>
<tr>
<td>chr12</td>
<td>53</td>
</tr>
<tr>
<td>chr13</td>
<td>65</td>
</tr>
<tr>
<td>chr14</td>
<td>74</td>
</tr>
<tr>
<td>chr15</td>
<td>59</td>
</tr>
<tr>
<td>chr16</td>
<td>48</td>
</tr>
<tr>
<td>chr17</td>
<td>47</td>
</tr>
<tr>
<td>chr18</td>
<td>43</td>
</tr>
<tr>
<td>chr19</td>
<td>19</td>
</tr>
<tr>
<td>chr2</td>
<td>87</td>
</tr>
<tr>
<td>chr3</td>
<td>62</td>
</tr>
<tr>
<td>chr4</td>
<td>57</td>
</tr>
<tr>
<td>chr5</td>
<td>52</td>
</tr>
<tr>
<td>chr6</td>
<td>82</td>
</tr>
<tr>
<td>chr7</td>
<td>51</td>
</tr>
<tr>
<td>chr8</td>
<td>57</td>
</tr>
<tr>
<td>chr9</td>
<td>69</td>
</tr>
<tr>
<td>chrMT</td>
<td>5</td>
</tr>
<tr>
<td>chrX</td>
<td>132</td>
</tr>
<tr>
<td>chrY</td>
<td>2</td>
</tr>
</tbody>
</table>

A subset of the intervals on a specific chromosome can be obtained using subsetting via `[]`.

> detail(exAI[seqnames(exAI) == "chrMT"])

<table>
<thead>
<tr>
<th>start</th>
<th>end</th>
<th>seq_name</th>
<th>strand</th>
<th>reads</th>
<th>matches</th>
<th>sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>964</td>
<td>986</td>
<td>chrMT</td>
<td>+</td>
<td>1</td>
<td></td>
<td>GTTTATGAGAGGAGATAAGTTGT</td>
</tr>
<tr>
<td>11613</td>
<td>11635</td>
<td>chrMT</td>
<td>+</td>
<td>10</td>
<td></td>
<td>AAGAAAGATTGCAAGAAACTGCTA</td>
</tr>
<tr>
<td>11613</td>
<td>11635</td>
<td>chrMT</td>
<td>+</td>
<td>1</td>
<td></td>
<td>AAGCAAGATTGCAAGAAACTGCTA</td>
</tr>
<tr>
<td>11613</td>
<td>11635</td>
<td>chrMT</td>
<td>+</td>
<td>1</td>
<td></td>
<td>AAGAAAGATTGCAAGAAACTGCTA</td>
</tr>
<tr>
<td>11613</td>
<td>11635</td>
<td>chrMT</td>
<td>+</td>
<td>1</td>
<td></td>
<td>AAGAAAGATTGCAAGAAACTGCTA</td>
</tr>
</tbody>
</table>

Finally, what is the number of aligned reads per chromosome?

> summary(exAI)

Number of aligned reads per chromosome:

<table>
<thead>
<tr>
<th>seqName</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>chr1</td>
<td>139</td>
</tr>
<tr>
<td>chr10</td>
<td>51</td>
</tr>
<tr>
<td>chr11</td>
<td>105</td>
</tr>
<tr>
<td>chr12</td>
<td>56</td>
</tr>
<tr>
<td>chr13</td>
<td>83</td>
</tr>
<tr>
<td>chr14</td>
<td>131</td>
</tr>
<tr>
<td>chr15</td>
<td>68</td>
</tr>
<tr>
<td>chr16</td>
<td>61</td>
</tr>
<tr>
<td>chr17</td>
<td>48</td>
</tr>
<tr>
<td>chr18</td>
<td>44</td>
</tr>
<tr>
<td>chr19</td>
<td>19</td>
</tr>
<tr>
<td>chr2</td>
<td>106</td>
</tr>
<tr>
<td>chr3</td>
<td>107</td>
</tr>
<tr>
<td>chr4</td>
<td>61</td>
</tr>
<tr>
<td>chr5</td>
<td>52</td>
</tr>
<tr>
<td>chr6</td>
<td>107</td>
</tr>
<tr>
<td>chr7</td>
<td>57</td>
</tr>
<tr>
<td>chr8</td>
<td>70</td>
</tr>
<tr>
<td>chr9</td>
<td>102</td>
</tr>
<tr>
<td>chrMT</td>
<td>14</td>
</tr>
<tr>
<td>chrX</td>
<td>206</td>
</tr>
<tr>
<td>chrY</td>
<td>2</td>
</tr>
</tbody>
</table>
2.4 Processing the aligned intervals

Sometimes, users may wish to combine certain aligned intervals. One intention could be to combine aligned reads at exactly the same position, which only differ in their sequence due to sequencing errors. Another objective could be to combine overlapping short reads that may be (degradation) products of the same primary transcript. The function `reduce` combines a set of aligned intervals into a single aligned interval, if the intervals:

- are on the same strand,
- are overlapping (or contained in each other) or directly adjacent to each other AND
- have the same read match specificity (see below)

**Read match specificity** By the read match specificity $r(I_i)$ of an interval $I_i$, we refer to the total number of valid alignments of reads that have been aligned to $I_i$, i.e. the total numbers of intervals with the same reads aligned in the whole genome (or other set of reference sequences). If $r(I_i) = 1$, the reads that were aligned to the interval $I_i$ have no other valid alignments in the whole genome, i.e. the interval $I_i$ is the unique match position of these reads. If $r(I_i) = 2$, the reads that were aligned to the interval $I_i$ have exactly one other valid alignment to another interval $I_j$, $j \neq i$. The match specificity is stored in the `matches` slot of objects of the class `AlignedGenomeIntervals`.

We first demonstrate the `reduce` using a toy example data object.

```r
> D <- AlignedGenomeIntervals(
+   start=c(1,3,4,5,8,10,11),
+   end=c(5,5,6,8,9,11,13),
+   chromosome=rep(c("chr1","chr2","chr3"), c(2,2,3)),
+   strand=c("-","-","+","+","+","+","+"),
+   sequence=c("ACATT","ACA","CGT","GTAA","AG","CT","TTT"),
+   reads=rep(1,7),
+   matches=c(rep(1,6),3))
> detail(D)
```

<table>
<thead>
<tr>
<th>start</th>
<th>end</th>
<th>seq_name</th>
<th>strand</th>
<th>reads</th>
<th>matches</th>
<th>sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>chr1</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>ACATT</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>chr1</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>ACA</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>chr2</td>
<td>+</td>
<td>1</td>
<td>1</td>
<td>CGT</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>chr2</td>
<td>+</td>
<td>1</td>
<td>1</td>
<td>GTAA</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>chr3</td>
<td>+</td>
<td>1</td>
<td>1</td>
<td>AG</td>
</tr>
<tr>
<td>6</td>
<td>11</td>
<td>chr3</td>
<td>+</td>
<td>1</td>
<td>1</td>
<td>CT</td>
</tr>
<tr>
<td>7</td>
<td>13</td>
<td>chr3</td>
<td>+</td>
<td>1</td>
<td>3</td>
<td>TTT</td>
</tr>
</tbody>
</table>

Calling the `reduce` method on these example data results in the following:

```r
> detail(reduce(D))
```

<table>
<thead>
<tr>
<th>start</th>
<th>end</th>
<th>seq_name</th>
<th>strand</th>
<th>reads</th>
<th>matches</th>
<th>sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>chr1</td>
<td>-</td>
<td>2</td>
<td>1</td>
<td>ACATT</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>chr2</td>
<td>+</td>
<td>2</td>
<td>1</td>
<td>CGTAA</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>chr3</td>
<td>+</td>
<td>2</td>
<td>1</td>
<td>AGCT</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>chr3</td>
<td>+</td>
<td>1</td>
<td>3</td>
<td>TTT</td>
</tr>
</tbody>
</table>
Note that the two last intervals still show overlap. However, the last interval is a non-unique match position of the respective reads (matches=3), in contrast to the other intervals.

Here is another example using the data introduced above.

```r
> S <- exAI[seqnames(exAI) == "chrX" & matches(exAI) == 1L & exAI[,1] > 1e8]
> detail(S)

    start end  seq_name strand reads matches sequence
 1 100768450 100768472 chrX  -   1      1       ATATAATACAACTGCTAAGGTT
 2 101311567 101311589 chrX  -   15      1       TGAGGTGTTGTACTGTGTTG
 3 101311567 101311589 chrX  -   12      1       TGAGGTGTTGTACTGTGTTG
 4 101311567 101311589 chrX  -   2       1       TGAGGTGTTGTACTGTGTTG
 5 101311567 101311589 chrX  -   1       1       TGAGGTGTTGTACTGTGTTG
 6 101311567 101311589 chrX  -   1       1       TGAGGTGTTGTACTGTGTTG
 7 148346896 148346918 chrX  +   4       1       TGAGGTAGTATGATAGTTG

Calling the `reduce` method on these data leads to the following result:

```r
detail(reduce(S))
```

    start end  seq_name strand reads matches sequence
 1 100768450 100768472 chrX  -   1       1       ATATAATACAACTGCTAAGGTT
 2 101311567 101311589 chrX  -   35      1       TGAGGTGTTGTACTGTGTTG
 3 148346896 148346918 chrX  +   4       1       TGAGGTAGTATGATAGTTG

Notice that the reads that match the same segment of the X chromosome differ in their last base. However, since most of the reads have a 'G' as final letter, the combined aligned interval also has a 'G' as the last letter.

The additional argument `method="exact"` can be specified if you want to solely combine intervals that have exactly the same start and stop position (but may have reads of slightly different sequence aligned to them). Consider the following example:

```r
> S2 <- exAI[seqnames(exAI) == "chr11" & matches(exAI) == 1L & exAI[,1] > 8e7]
> detail(S2)

    start end  seq_name strand reads matches sequence
 1 86397621 86397643 chr11  -   20      1       TAGCTTATCAGACTGTGATTTAC
 2 86397621 86397643 chr11  -   1       1       TAGCTTATCAGACTGTGATTTAC
 3 86397621 86397643 chr11  -   2       1       TAGCTTATCAGACTGTGATTTAC
 4 88515338 88515360 chr11  -   1       1       GGTGCAGGGAGCGCCAGTGTCTC
 5 96178500 96178522 chr11  +   2       1       TACCCTGTAGATCCGAATTTTG
 6 96178501 96178523 chr11  +   1       1       ACCCTGTAGATCCGAATTTTG
 7 108873196 108873218 chr11  -   1       1       AGTGCAGGTAACGGACCCGCTACC
```

```r
> detail(reduce(S2, method="exact"))
```

```
Notice that the 6th aligned interval in S2 is only shifted by 1 nt from the 5th one. By default, the function reduce would merge them into one aligned genome interval. However, when method="exact" is specified, these two intervals are not merged since they are not at exactly the same position. There are additional methods for restricting the merging to intervals with the same 5’- and 3’-ends (specify method="same5" and method="same3", respectively).

2.5 Visualising the aligned genome intervals

The package girafe contains functions for visualising genomic regions with aligned reads.

> plot(exAI, mgi.gi, chr="chrX", start=50400000, +    end=50410000, show="minus")

See the result in Figure 1.

![Chromosome chrX coordinate [bp]](image)

Figure 1: A 10-kb region on the mouse X chromosome. Reads aligned to the Watson strand in this region would be shown above the chromosome coordinate axis with the annotation of genome elements in this region, while reads aligned to the Crick strand are shown below. In the region shown, there are only intervals with aligned reads on the Crick strand, and these four intervals overlap with annotated microRNA positions.

Note that the annotation of genome elements (as shown in Figure 1) has to be supplied to the function. Here the object mgi.gi contains most annotated genes and ncRNAs for the
mouse genome (assembly: \textit{mm9}). This object has been created beforehand\textsuperscript{2} and it is of class \textit{Genome\_intervals\_stranded}, a class defined in package \textit{genomeIntervals}.

### 2.6 Summarising the data using sliding windows

The data can be searched for regions of defined interest using a sliding-window approach implemented in the function \texttt{perWindow}. For each window, the number of intervals with aligned reads, the total number of reads aligned, the number of unique reads aligned, the fraction of intervals on the Plus-strand, and the higher number of aligned reads at a single interval within the window are reported.

```r
> exPX <- perWindow(exAI, chr="chrX", winsize=1e5, step=0.5e5)
> head(exPX[order(exPX$n.overlap, decreasing=TRUE),])
```

<table>
<thead>
<tr>
<th>chr</th>
<th>start</th>
<th>end</th>
<th>n.overlap</th>
<th>n.reads</th>
<th>n.unique</th>
<th>frac.plus</th>
<th>max.reads</th>
</tr>
</thead>
<tbody>
<tr>
<td>chrX</td>
<td>50341103</td>
<td>50441102</td>
<td>18</td>
<td>55</td>
<td>18</td>
<td>0</td>
<td>28</td>
</tr>
<tr>
<td>chrX</td>
<td>50391103</td>
<td>50491102</td>
<td>18</td>
<td>55</td>
<td>18</td>
<td>0</td>
<td>28</td>
</tr>
<tr>
<td>chrX</td>
<td>101241103</td>
<td>101341102</td>
<td>5</td>
<td>34</td>
<td>5</td>
<td>0</td>
<td>18</td>
</tr>
<tr>
<td>chrX</td>
<td>101291103</td>
<td>101391102</td>
<td>5</td>
<td>34</td>
<td>5</td>
<td>0</td>
<td>18</td>
</tr>
<tr>
<td>chrX</td>
<td>64041103</td>
<td>64141102</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>chrX</td>
<td>63991103</td>
<td>64091102</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>first</th>
<th>last</th>
</tr>
</thead>
<tbody>
<tr>
<td>50401220</td>
<td>50407226</td>
</tr>
<tr>
<td>50401220</td>
<td>50407226</td>
</tr>
<tr>
<td>101311567</td>
<td>101311589</td>
</tr>
<tr>
<td>101311567</td>
<td>101311589</td>
</tr>
<tr>
<td>64049984</td>
<td>64092296</td>
</tr>
<tr>
<td>64049984</td>
<td>64067192</td>
</tr>
</tbody>
</table>

### 2.7 Exporting the data

The package \textit{girafe} also contains methods for exporting the data into tab-delimited text files, which can be uploaded to the UCSC genome browser\textsuperscript{3} as ‘custom tracks’.

Currently, there are methods for exporting the data in ‘bed’ format and ‘bedGraph’ format, either writing intervals from both strands into one file or into two separate files. Details about these track formats can be found at the UCSC genome browser web pages.

```r
> export(exAI, con="export.bed",
+ format="bed", name="example_reads",
+ description="Example reads",
+ color="100,100,255", visibility="pack")
```

\textsuperscript{2}See the script \texttt{prepareAnnotation.R} in the package scripts directory for an example of how to create such an object.

\textsuperscript{3}\url{http://genome.ucsc.edu}
Additional arguments to the export function, besides `object`, `con`, and `format`, are treated as attributes for the track definition line, which specifies details concerning how the data should be visualised in the genome browser.

Users may also wish to consult the Bioconductor package `rtracklayer` for data transfer and direct interaction between R and the UCSC genome browser.

### 2.8 Overlap with annotated genome features

Next, we determine the degree of overlap of the aligned reads with annotated genomic elements. In this example, the annotated genome elements are provided as an object of class `Genome_intervals_stranded`\(^4\). This objects needs to be created beforehand. See the script `prepareAnnotation.R` in the package scripts directory\(^5\) for an example of how to create such an object.

```r
> exOv <- interval_overlap(exAI, mgi.gi)
```

How many elements do read match positions generally overlap?

```r
> table(listLen(exOv))
```

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>815</td>
<td>340</td>
<td>130</td>
<td>1</td>
</tr>
</tbody>
</table>

What are the 12 elements overlapped by a single match position?

```r
> mgi.gi$ID[exOv[[which.max(listLen(exOv))]]]
```

[1] "Pcdha1" "Pcdha2" "Pcdha3" "Pcdha4" "Pcdha5" "Pcdha6" "Pcdha7"

[8] "Pcdha8" "Pcdha9" "Pcdha10" "Pcdha11" "Pcdha12"

And in general, what kinds of annotated genome elements are overlapped by reads?

```r
> (tabOv <- table(as.character(mgi.gi$type)[unlist(exOv)]))
```

| gene lincRNA miRNA ncRNA pseudogene rRNA snoRNA tRNA |
|--------|--------|--------|--------|---------|--------|---------|
| 238    | 15     | 297    | 19     | 28      | 10     | 1       |

We display these overlap classes using a pie chart.

```r
> my.cols <- brewer.pal(length(tabOv), "Set3")
> pie(tabOv, col=my.cols, radius=0.88)
```
Figure 2: Pie chart showing what kinds of genome elements are overlapped by aligned reads. Note that the proportions of the pie chart are given by the proportions among all annotated genome elements that have ≥ 1 reads mapped to them and not by the total numbers of reads mapped to elements of that class, in which case the proportion of the miRNA class would be significantly larger.

See the result in Figure 2.

Note that function `interval.overlap` only determines whether two intervals are overlapping by at least one base. For restricting the result to intervals overlapping by at least a certain number of bases or by a fraction of the interval’s length, see the function `fracOverlap`.

3 Memory usage

At the moment, `girafe` and the packages that it depends on, retain all the information concerning the read alignments in memory. This allows quick access to and swift operations on the data, but may limit the package’s usability on machines with low amounts of RAM.

The step with the highest RAM requirements is importing the alignments and saving them as objects of the `AlignedRead` class using the functionality in package `ShortRead`. Usually, objects of the `AlignedGenomeIntervals` class are created starting from `AlignedRead` objects and the `AlignedRead` objects can safely be discarded after this step. Since the data is summarised in that process, `AlignedGenomeIntervals` objects require about 10–100 times

---

Notes:

4 Objects of class `Genome_intervals` and `AlignedGenomeIntervals` are also allowed.
5 `system.file("scripts", package="girafe")`
less memory than the original \textit{AlignedRead} object\textsuperscript{6}. We recommend that the import of the alignments and the generation of the \textit{AlignedGenomeIntervals} are performed using a separate script which only needs to be called once on a machine with sufficient RAM.

A suggestion for limiting memory usage is to perform the read alignments and import of the results in batches of a few million reads each. The batch-wise result \textit{AlignedGenomeIntervals} objects can later be combined using the basic R function “c”, the standard way of combining objects, optionally followed by calls of the \texttt{reduce} function.

For alignments in SAM/BAM format, the \textit{Samtools} software suite (Li et al., 2009) as well as the Bioconductor package \textit{Rsamtools} allow the user to access and import only selected subsets of the data, which also leads to a lower memory footprint. For details, please refer to the documentation of these packages.

Finally, while the processing of \textit{AlignedRead} objects is the principal way of generating \textit{AlignedGenomeIntervals} objects, there is also a convenience function called \textit{AlignedGenomeIntervals}, which can be used to create these objects from simpler objects in the work space, such as data read in using basic R functions such as \texttt{scan}. This convenience function may be easier to use for importing and processing the data in manageable chunks.

When following these suggestions, most operations with the \textit{girafe} package should be possible on a machine with 4 Gb of RAM, and we have not so far encountered a situation that requires more than 12 Gb (state as of the end of 2009). However, increased throughput of sequencing machines and longer reads will lead to increased memory requirements. Future developments of this and other NGS-related Bioconductor packages will therefore likely concern ways to reduce the memory footprint. One idea is to make use of packages like \texttt{ff}, which provide ways of swapping data from RAM to flat files on the hard disk, while still allowing fast and direct access to the data.

\section{A word about speed}

For improving the run time on machines with multiple processors, some of the functions in the \textit{girafe} package have been implemented to make use of the functionality in the \texttt{parallel} package. If \texttt{parallel} has been attached and initialised before calling these functions, the functions will make use of \texttt{mclapply} instead of the normal \texttt{lapply} function. The number of cores to be used in parallel is determined by the \texttt{mc.cores} option (see the example below).

For example, if \texttt{parallel} is functional on a given system\textsuperscript{7}, there should be an obvious speed improvement in the following code example.

\begin{verbatim}
> library("parallel")
> options("mc.cores"=4) # adjust to your machine
> exAI.R <- reduce(exAI, mem.friendly=TRUE)
\end{verbatim}

\textsuperscript{6}e.g., an \textit{AlignedRead} object for holding $10^6$ reads of length 36 bp aligned to the mouse reference genome occupies about 1.4 Gb in RAM but is processed into an \textit{AlignedGenomeIntervals} object of size 66.7 Mb

\textsuperscript{7}The \texttt{mclapply} function currently does not support Windows operating systems.
5 Links to other Bioconductor packages

The girafe package is mostly built upon the interval notation and implementation provided by the packages intervals and genomeIntervals. Functions from the ShortRead package (Morgan et al., 2009) are used for importing the data, and Biostrings provides help for working with the read nucleotide sequences. girafe also makes limited use of the Rle and IRanges classes defined in the IRanges package. Furthermore, the data can be converted between object classes defined in girafe and IRanges.

We note that many of the interval operations in girafe can also be performed using classes and functions defined in the IRanges package. However, the scope of the packages is slightly different. While IRanges is meant to be a generic infrastructure package of the Bioconductor project, the aim of girafe is to provide a single, comparatively lightweight, object class for working with reads aligned to the genome, the AlignedGenomeIntervals. This class and its methods allow easy access to such data and facilitate standard operations and workflows.

There is some overlap in functionality between girafe, IRanges, GenomicRanges and track-layer. The range of interactions between these packages and new Bioconductor packages related to next-generation sequencing is likely to increase over the releases. Our aim is to provide users with a broad range of alternatives for selecting the classes and functions that are most suited for their workflows.

Package versions

This vignette was generated using the following package versions:

- R version 3.4.0 (2017-04-21), x86_64-pc-linux-gnu
- Running under: Ubuntu 16.04.2 LTS
- Matrix products: default
- BLAS: /home/biocbuild/bbs-3.5-bioc/R/lib/libRblas.so
- LAPACK: /home/biocbuild/bbs-3.5-bioc/R/lib/libRlapack.so
- Base packages: base, datasets, grDevices, graphics, grid, methods, parallel, stats, stats4, utils
- Other packages: AnnotationDbi 1.38.0, Biobase 2.36.0, BiocGenerics 0.22.0, BiocParallel 1.10.0, Biostrings 2.44.0, DelayedArray 0.2.0, GenomeInfoDb 1.12.0, GenomicAlignments 1.12.0, GenomicRanges 1.28.0, IRanges 2.10.0, RColorBrewer 1.1-2, Rsamtools 1.28.0, S4Vectors 0.14.0, ShortRead 1.34.0, SummarizedExperiment 1.6.0, XVector 0.16.0, genomeIntervals 1.32.0, girafe 1.28.0, intervals 0.15.1, matrixStats 0.52.2, org.Mm.eg.db 3.4.1
- Loaded via a namespace (and not attached): DBI 0.6-1, GenomeInfoDbData 0.99.0, Matrix 1.2-9, RCurl 1.95-4.8, RSQLite 1.1-2, Rcpp 0.12.10, bitops 1.0-6, compiler 3.4.0, digest 0.6.12, hwriter 1.3.2, lattice 0.20-35, latticeExtra 0.6-28, memoise 1.1.0, tools 3.4.0, zlibbioc 1.22.0

Acknowledgements

Many thanks to Nicolas Servant, Valérie Cognat, Nicolas Delhomme, and especially Patrick Aboyoun for suggestions and feedback on the package. Special thanks to Julien Gagneur and Richard Bourgon
for writing genomeIntervals and for rapidly answering all my questions regarding the package. The plotting functions in package girafe are largely based on the function plotAlongChrom and its auxiliary functions from package tilingArray, most of which were written by Wolfgang Huber.

**Funding:** This work was supported by the Institut Curie, INCa "GepiG".

## References


