An overview of the *girafe* package

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

October 24, 2023

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 *ShortRead* (Morgan et al., 2009) and *genomeIntervals*.

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

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

- J Toedling, C Ciaudo, O Voinnet, E Heard and E Barillot (2010) girafe – an R/Bioconductor package for functional exploration of aligned next-generation sequencing reads. *Bioinformatics*, 26(22):2902-3.

Getting help

If possible, please send questions about *girafe* to the Bioconductor mailing list.
See `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).

[1] The package has been developed for analysing single-end reads (fragment libraries) and does not support mate-pair reads yet.

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

2.1 Adapter trimming

We load reads that include parts of the adapter sequence.

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

```
> 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.

```
> adapter <- "CTGTAGGCACCATCAAT"
> 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*.

```
> 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 *girafe* 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))
```

```
 chr1 chr10 chr11 chr12 chr13 chr14 chr15 chr16 chr17 chr18 chr19  chr2  chr3
  112    50    60    53    65    74    59    48    47    43    19    87    62
 chr4  chr5  chr6  chr7  chr8  chr9 chrMT  chrX  chrY
   57    52    82    51    57    69     5   132     2
```

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

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

```
  start   end seq_name strand reads matches                sequence
1   964   986    chrMT      +     1       1 GTTTATGAGAGGAGATAAGTTGT
2 11613 11635    chrMT      +    10       2 AAGAAAGATTGCAAGAACTGCTA
3 11613 11635    chrMT      +     1       2 AAGCAAGATTGCAAGAACTGCTA
4 11613 11635    chrMT      +     1       2 AAGAACGATTGCAAGAACTGCTA
5 11613 11635    chrMT      +     1       3 AAGAAAGATTGCAAGAACTGTTA
```

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

```
> summary(exAI)
```

```
Number of aligned reads per chromosome:
 chr1 chr10 chr11 chr12 chr13 chr14 chr15 chr16 chr17 chr18 chr19  chr2  chr3
  139    51   105    56    83   131    68    61    48    44    19   106   107
 chr4  chr5  chr6  chr7  chr8  chr9 chrMT  chrX  chrY
   61    52   107    57    70   102    14   206     2
```

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.

```
> 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)
```

```
  start end seq_name strand reads matches sequence
1     1   5     chr1      -     1       1    ACATT
2     3   5     chr1      -     1       1      ACA
```

```
3     4   6     chr2      +     1       1      CGT
4     5   8     chr2      +     1       1     GTAA
5     8   9     chr3      +     1       1       AG
6    10  11     chr3      +     1       1       CT
7    11  13     chr3      +     1       3      TTT
```

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

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

```
  start end seq_name strand reads matches sequence
1     1   5     chr1      -     2       1    ACATT
2     4   8     chr2      +     2       1    CGTAA
3     8  11     chr3      +     2       1     AGCT
4    11  13     chr3      +     1       3      TTT
```

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.

```
> 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 ATATAATACAACCTGCTAACTGT
2 101311567 101311589     chrX      -    18       1 TGAGGTTGGTGTACTGTGTGTGG
3 101311567 101311589     chrX      -    12       1 TGAGGTTGGTGTACTGTGTGTGA
4 101311567 101311589     chrX      -     2       1 TGAGGTTGGTGTACTGTGTGTGT
5 101311567 101311589     chrX      -     1       1 TGACGTTGGTGTACTGTGTGTGA
6 101311567 101311589     chrX      -     1       1 TGAGGTTGGTGTACTGTGTGCGG
7 148346896 148346918     chrX      +     4       1 TGAGGTAGTAGATTGTATAGTTT
```

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

```
> detail(reduce(S))
```

```
      start       end seq_name strand reads matches                sequence
1 100768450 100768472     chrX      -     1       1 ATATAATACAACCTGCTAACTGT
2 101311567 101311589     chrX      -    34       1 TGAGGTTGGTGTACTGTGTGTGG
3 148346896 148346918     chrX      +     4       1 TGAGGTAGTAGATTGTATAGTTT
```

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 als 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:

```
> 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 TAGCTTATCAGACTGATGTTTAC
2  86397621  86397643    chr11      -     1       1 TAGATTATCAGACTGATGTTTAC
3  86397621  86397643    chr11      -     2       1 TAGCTTATCAGACTGATGTTCAC
4  88515338  88515360    chr11      -     1       1 GGTGCAGGGAGCGCCAGTGTCTC
5  96178500  96178522    chr11      +     2       1 TACCCTGTAGATCCGAATTTTTG
6  96178501  96178523    chr11      +     1       1 ACCCTGTAGATCCGAATTTGTGA
7 108873196 108873218    chr11      -     1       1 AGTGCGGTAACGCGACCGCTACC
```

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

```
      start       end seq_name strand reads matches                 sequence
1  86397621  86397643    chr11      -    23       1 TAGCTTATCAGACTGATGTTTAC
2  88515338  88515360    chr11      -     1       1 GGTGCAGGGAGCGCCAGTGTCTC
3  96178500  96178522    chr11      +     2       1 TACCCTGTAGATCCGAATTTTTG
4  96178501  96178523    chr11      +     1       1 ACCCTGTAGATCCGAATTTGTGA
5 108873196 108873218    chr11      -     1       1 AGTGCGGTAACGCGACCGCTACC
```

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.

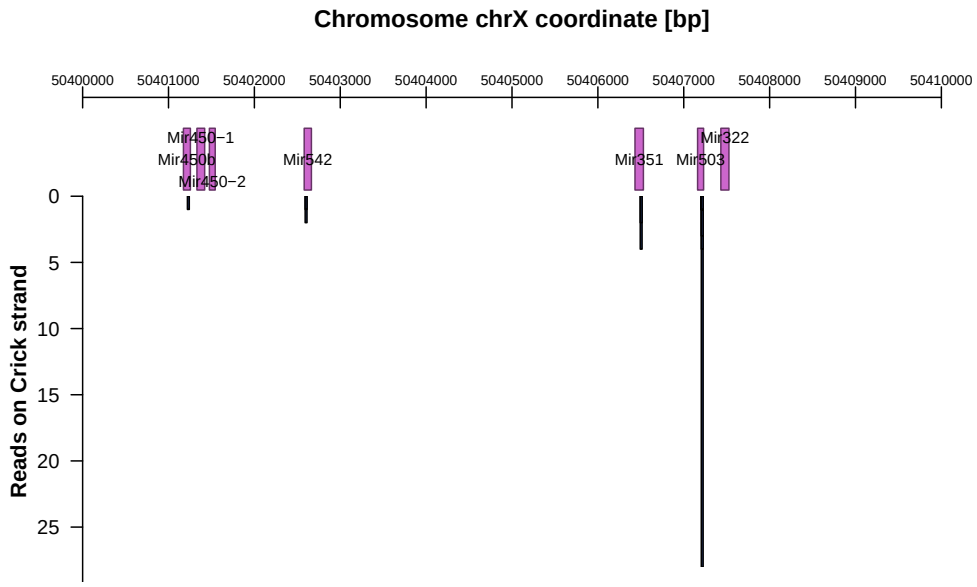

Figure 1: *A 10-kb region on the mouse X chromosome. Reads aligned to the Watson strand in this region would be shown above the chromosome coordinate axis with the annotation of genome elements in this region, while reads aligned to the Crick strand are shown below. In the region shown, there are only 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: *mm9*). This object has been created beforehand[2] and it is of class *Genome_intervals_stranded*, a class defined in package *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 `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.

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

```
      chr     start       end n.overlap n.reads n.unique frac.plus max.reads
942   chrX  50341103  50441102        18      55       18         0        28
```

[2]See the script `prepareAnnotation.R` in the package scripts directory for an example of how to create such an object.

```
943  chrX  50391103  50491102        18       55       18         0        28
1960 chrX 101241103 101341102         5       34        5         0        18
1961 chrX 101291103 101391102         5       34        5         0        18
1216 chrX  64041103  64141102         4        5        4         0         2
1215 chrX  63991103  64091102         3        4        3         0         2
         first      last
942   50401220  50407226
943   50401220  50407226
1960 101311567 101311589
1961 101311567 101311589
1216  64049984  64092296
1215  64049984  64067192
```

2.7 Exporting the data

The package *girafe* also contains methods for exporting the data into tab-delimited text files, which can be uploaded to the UCSC genome browser[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.

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

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.

[3] http://genome.ucsc.edu

[4] Objects of class *Genome_ intervals* and *AlignedGenomeIntervals* are also allowed.

[5] `system.file("scripts", package="girafe")`

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

How many elements do read match positions generally overlap?

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

```
  0   1   2  12
815 340 130   1
```

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

```
> 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?

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

```
      gene    lincRNA      miRNA      ncRNA pseudogene       rRNA     snoRNA
       238         15        297         19         28         10          1
      tRNA
         4
```

We display these overlap classes using a pie chart.

```
> my.cols <- brewer.pal(length(tabOv), "Set3")
> pie(tabOv, col=my.cols, radius=0.88)
```

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`.

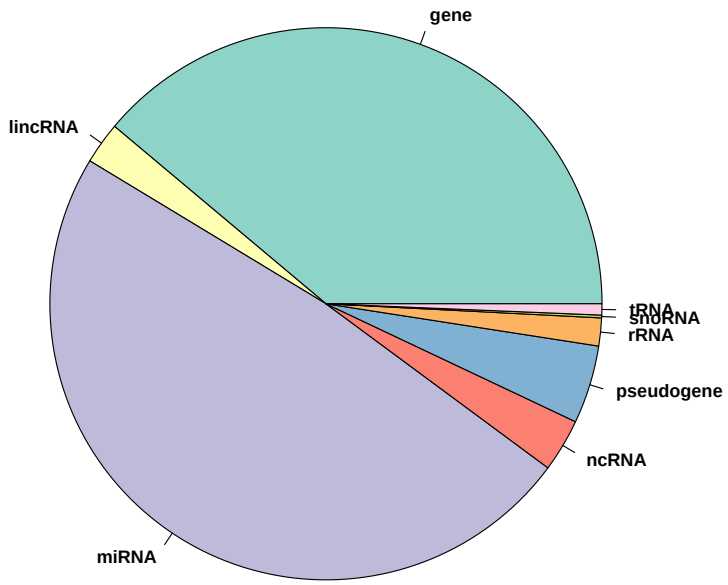

Figure 2: *Pie chart showing what kinds of genome elements are overlapped by aligned reads. Note that the proportions of the pie chart are given by the proportions among all annotated genome elements that have* ≥ 1 *reads mapped to them and not by the total numbers of reads mapped to elements of that class, in which case the proportion of the miRNA class would be significantly larger.*

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 less memory than the original *AlignedRead* object[6]. We recommend that the import of the alignments and the generation of the *AlignedGenomeIntervals* are performed using a separate script which only needs to be called once on a machine with sufficient RAM.

[6] e.g., an *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 *AlignedGenomeIntervals* object of size 66.7 Mb

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 *AlignedGenomeIntervals* objects can later be combined using the basic R function "c", the standard way of combining objects, optionally followed by calls of the `reduce` function.

For alignments in SAM/BAM format, the *Samtools* software suite (Li et al., 2009) as well as the Bioconductor package *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 *AlignedRead* objects is the principal way of generating *AlignedGenomeIntervals* objects, there is also a convenience function called
`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 `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 *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 *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.

4 A word about speed

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

For example, if *parallel* is functional on a given system[7], there should be an obvious speed improvement in the following code example.

```
> library("parallel")
> options("mc.cores"=4) # adjust to your machine
> exAI.R <- reduce(exAI, mem.friendly=TRUE)
```

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.,

[7] The `mclapply` function currently does not support Windows operating systems.

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 *tracklayer*. 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 Under development (unstable) (2023-10-22 r85388), `x86_64-pc-linux-gnu`
- Running under: `Ubuntu 22.04.3 LTS`
- Matrix products: default
- BLAS: `/home/biocbuild/bbs-3.19-bioc/R/lib/libRblas.so`
- LAPACK: `/usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.10.0`
- Base packages: base, datasets, graphics, grDevices, grid, methods, stats, stats4, utils
- Other packages: AnnotationDbi 1.65.0, Biobase 2.63.0, BiocGenerics 0.49.0, BiocParallel 1.37.0, Biostrings 2.71.0, GenomeInfoDb 1.39.0, genomeIntervals 1.59.0, GenomicAlignments 1.39.0, GenomicRanges 1.55.0, girafe 1.55.0, intervals 0.15.4, IRanges 2.37.0, MatrixGenerics 1.15.0, matrixStats 1.0.0, org.Mm.eg.db 3.18.0, RColorBrewer 1.1-3, Rsamtools 2.19.0, S4Vectors 0.41.0, ShortRead 1.61.0, SummarizedExperiment 1.33.0, XVector 0.43.0
- Loaded via a namespace (and not attached): abind 1.4-5, bit 4.0.5, bit64 4.0.5, bitops 1.0-7, blob 1.2.4, cachem 1.0.8, cli 3.6.1, codetools 0.2-19, compiler 4.4.0, crayon 1.5.2, DBI 1.1.3, DelayedArray 0.29.0, deldir 1.0-9, fastmap 1.1.1, GenomeInfoDbData 1.2.11, httr 1.4.7, hwriter 1.3.2.1, interp 1.1-4, jpeg 0.1-10, KEGGREST 1.43.0, lattice 0.22-5, latticeExtra 0.6-30, Matrix 1.6-1.1, memoise 2.0.1, parallel 4.4.0, pkgconfig 2.0.3, png 0.1-8, R6 2.5.1, Rcpp 1.0.11, RCurl 1.98-1.12, rlang 1.1.1, RSQLite 2.3.1, S4Arrays 1.3.0, SparseArray 1.3.0, tools 4.4.0, vctrs 0.6.4, zlibbioc 1.49.0

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

R. Edgar, M. Domrachev, and A. E. Lash. Gene Expression Omnibus: NCBI gene expression and hybridization array data repository. *Nucleic Acids Res*, 30(1):207–210, Jan 2002.

R. C. Gentleman, V. J. Carey, D. J. Bates, B. M. Bolstad, M. Dettling, S. Dudoit, B. Ellis, L. Gautier, Y. Ge, J. Gentry, K. Hornik, T. Hothorn, W. Huber, S. Iacus, R. Irizarry, F. Leisch, C. Li, M. Maechler, A. J. Rossini, G. Sawitzki, C. Smith, G. K. Smyth, L. Tierney, Y. H. Yang, and J. Zhang. Bioconductor: Open software development for computational biology and bioinformatics. *Genome Biology*, 5:R80, 2004.

B. Langmead, C. Trapnell, M. Pop, and S. L. Salzberg. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology*, 10(3):R25, 2009.

H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, and R. D. and. The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, 25(16):2078–9, Aug 2009.

M. Morgan, S. Anders, M. Lawrence, P. Aboyoun, H. Pages, and R. Gentleman. ShortRead: a Bioconductor package for input, quality assessment and exploration of high-throughput sequence data. *Bioinformatics*, 25(19):2607–2608, Oct 2009.

O. H. Tam, A. A. Aravin, P. Stein, A. Girard, E. P. Murchison, S. Cheloufi, E. Hodges, M. Anger, R. Sachidanandam, R. M. Schultz, and G. J. Hannon. Pseudogene-derived small interfering RNAs regulate gene expression in mouse oocytes. *Nature*, 453(7194):534–538, May 2008.

J. Toedling, C. Ciaudo, O. Voinnet, E. Heard, and E. Barillot. girafe - an R/Bioconductor package for functional exploration of aligned next-generation sequencing reads. *Bioinformatics*, 26(22):2902–2903, Nov 2010.