Chapter 16

Visualizing Genomic Data Using Gviz and Bioconductor

Florian Hahne and Robert Ivanek

Abstract

The *Gviz* package offers a flexible framework to visualize genomic data in the context of a variety of different genome annotation features. Being tightly embedded in the Bioconductor genomics landscape, it nicely integrates with the existing infrastructure, but also provides direct data retrieval from external sources like Ensembl and UCSC and supports most of the commonly used annotation file types. Through carefully chosen default settings the package greatly facilitates the production of publication-ready figures of genomic loci, while still maintaining high flexibility due to its ample customization options.

Key words Visualization, Genomics, NGS, Annotation

1 Introduction

A typical genomics experiments these days produces massive amounts of data, which usually do not lend themselves to direct visual inspection. However, there is an intermediate level of processing that generates useful summaries, allowing to combine many different pieces of genomic information in a single graph. These visualizations are typically alignments of genomic features on a common horizontal scale, including such diverse feature types as gene or transcript models, CpG islands, transcription factor binding sites or other regulatory elements, chromosomal staining bands, Next Generation Sequencing read, ChIP measurements, and many more. Plotting these features together with the derived numerical data has proven to be tremendously useful for putting experimental results in context with genomic loci and to derive hypotheses. Relevant annotation data may be hosted in public repositories like the NCBI, Ensembl or at UCSC, or it may have been previously generated in-house. Many of the currently available genome browsers do a reasonable job in displaying genome annotation data, and there are options to connect to some of them from within R, for instance via the *rtracklayer* package [1]. However, none of these solutions offer the flexibility of the full R graphics

Ewy Mathé and Sean Davis (eds.), *Statistical Genomics: Methods and Protocols*, Methods in Molecular Biology, vol. 1418, DOI 10.1007/978-1-4939-3578-9_16,

system to display large numeric data in a multitude of different ways. The *Gviz* package aims to close this gap by providing a structured visualization framework to plot any type of data along genomic coordinates. It is loosely based on the *GenomeGraphs* package [2], however the complete class hierarchy as well as all the plotting methods have been restructured in order to increase performance and flexibility.

The fundamental concept behind the *Gviz* package is similar to the approach taken by most other genome browsing tools, in that individual types of genomic features or data are represented by separate tracks. A track in this context is really just a section within a composite visualization with a fixed horizontal scale. In the case of numeric data it may also contain a vertical scale to display their magnitude, however there are also more qualitative annotation tracks like transcript models for which the vertical axis does not convey any particular meaning. Within the *Gviz* package, each track constitutes a single object inheriting from a common base class, and there are constructor functions as well as a broad range of methods to instantiate, to interact with, and to plot these objects. When combining multiple track objects in a composite graph, the individual tracks will always share the same genomic coordinate system on the horizontal axis, thus taking the burden of aligning the individual elements from the user. A powerful customization interface provides ample opportunity to fine-tune almost every aspect of the visualization, while at the same time tries hard to come up with both appealing and meaningful default settings. In the remainder of this short article we want to highlight some of the features of the *Gviz* package in a real-world sample data set.

2 Materials

2.1 How to Install

Like any other Bioconductor package, *Gviz* should be installed using the *biocLite()* function. Mainly owing to its versatility it comes with quite a number of other package dependencies. The package can be loaded using the *library()* function.

```
source("http://bioconductor.org/biocLite.R")
biocLite("Gviz")
library(Gviz)
```

2.2 Data Set Description

For the purpose of this demonstration we will use a dataset from an RNA-seq experiment in which the authors studied the effect of RNAi depletion of the *Pasilla* (*PS*) gene on the transcriptome. *PS* is a known splice regulator and it is the *Drosophila melanogaster* ortholog of the mammalian RNA binding proteins *NOVA1* and *NOVA2* [3]. The authors used RNA interference to knock down the *PS* gene in S2-DRSC cells and compared their overall gene expression levels upon *PS* depletion to the expression in untreated cells. The experiment was performed with three biological replicates per condition. The data were deposited in GEO under the submission numbers *GSM461176-GSM461181*. Preprocessed data from this experiment are also directly available as the Bioconductor data package *pasilla*. In this data set, the original RNASeq reads were aligned using Tophat version 1.2.0 [4] with default parameters against the reference Drosophila melanogaster genome (version BDGP5.25.62 from Ensembl [5]). Similar as before, the *pasilla* data package can be installed using the *biocLite()* function and loaded using the *library()* function.

3 Methods

3.1 Track Objects

Before diving into the real world example we need to get a basic understanding of the fundamental concepts in the *Gviz* package, and to that end we first create the most simple track element there is: a genomic coordinates axis indicator. As briefly mentioned before, tracks within *Gviz* are represented by S4 objects inheriting from a common base class called *GdObject*. For the casual use it is not important to know all the details of this class. It mainly adds infrastructure for the very basic operations on a *Gviz* track like plotting and customization. The specific functionality of a particular track type is provided through a more dedicated child class and its methods. For our initial genome axis example we will need to instantiate an object of class *GenomeAxisTrack*. Conveniently, there is a constructor function available for this task with the same name.

```
axTrack <- GenomeAxisTrack() axTrack
## Genome axis 'Axis'
```

With our first track object being created we may now proceed to the plotting. There is a single function *plotTracks()* that handles all of this. As we will learn in the remainder of this chapter, *plotTracks()* is quite powerful and has a number of very useful additional arguments. For now we will keep things simple and just plot the single axis track using the default settings. The only two additional arguments we have to pass on are the start and the end coordinates of the plotting range.

```
plotTracks(axTrack, from=1e6, to=10e6)
```

The default values that have been chosen by the *Gviz* package for drawing the track object in Fig. 1 usually result in both visually

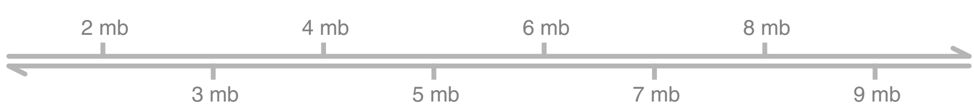

Fig. 1 A simple genome axis annotation track

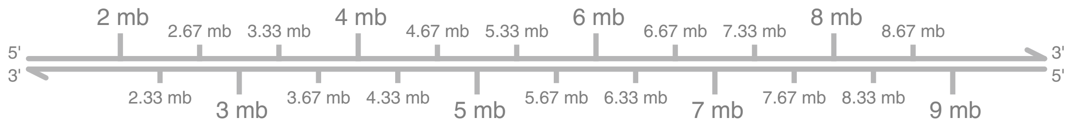

Fig. 2 A genome axis annotation track with some custom settings: direction indicators at and more fine-grained axis tick marks

appealing and also meaningful plots. However, visualization is an art in itself, and no software can ever be smart enough to guess the exact intention of a plot. To this end, *Gviz* offers a powerful customization framework to enable tweaking pretty much every aspect of the plot. We will discuss this in more detail in Subheading 3.2 below. As a simple example, we could decorate our genome axis with direction indicators and add additional tick marks at shorter intervals, as shown in Fig. 2.

```
plotTracks(axTrack, from=1e6, to=10e6, add53=TRUE, add35=
TRUE, littleTicks=TRUE)
```

So far we have not really plotted any genomic annotation data. One potentially interesting application would be to visualize the presence of repeats on a single chromosome. These data can be conveniently described by simple run-length encoding along genomic coordinates. All we need are the start and end positions for the repeats, and the chromosome to which these coordinates refer. The *Gviz* package provides the *AnnotationTrack* class to deal with exactly this sort of data. Its constructor function *AnnotationTrack ()* takes a variety of different inputs to digest the coordinate information, including *GRanges* objects, which are the recommended containers for genomic run-length encoded data in the Bioconductor world. However one can also create an *AnnotationTrack* object from quite generic inputs like *data.frames*. For this simple example we will download a RepeatMasker track for a single *Drosophila Melanogaster* chromosome and turn its first 50 entries into an *AnnotationTrack* object.

```
url <- "http://hgdownload.cse.ucsc.edu/goldenPath/dm3/database/chr3R_rmsk.
                    txt.gz"
con <- gzcon(url(url, open="r"))
repeats <- read.table(textConnection(readLines(con)), nrow=50)[, 6:8]
annoTrack <- AnnotationTrack(start=repeats[,2],
                    end=repeats[,3], chromosome=repeats[,1],
                            genome="dm3", name="Repeats", stacking="dense")
                    annoTrack
## AnnotationTrack 'Repeats'
## | genome: dm3
## | active chromosome: chr3R
## | annotation features: 50
```

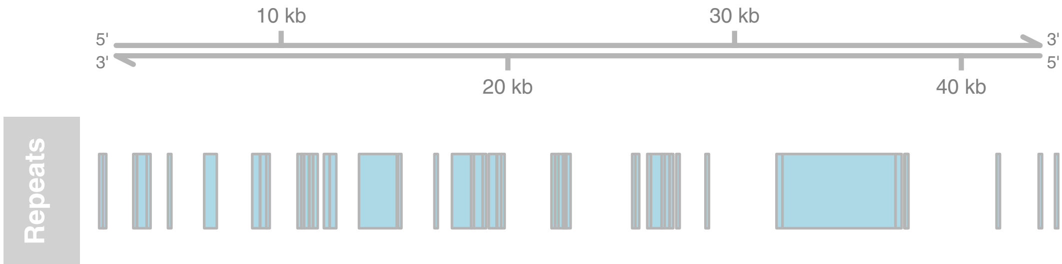

Fig. 3 Annotation track showing the distribution of the first 50 repeat elements on the *Drosophila melanogaster* 3R chromosome

Track objects in isolation are not particularly helpful, and we usually want to combine multiple tracks in a composite plot. In order to plot our new *AnnotationTrack* together with the *GenomeAxisTrack* from before, we need to provide them in the form of a list to the *plotTracks()* function (Fig. 3). We do not really have to explicitly provide the plotting ranges this time, since the *Gviz* package will try to extract the most extreme coordinates from the provided track objects if possible. Because the horizontal scales in all the individual tracks are synchronized, we automatically get the correct alignment of all the displayed features. We also see that the annotation track has been decorated with a title panel, displaying its name. Since names do not make too much sense for the genome axis, the title panel is hidden by default for that track. Again, this is an example where the package tries hard to select meaningful default settings. Later in this chapter we will see that the title panel is used to convey other information as well by some of the more complex track classes.

```
plotTracks(list(axTrack, annoTrack))
```

3.2 Display Parameters

Customization of track objects in the *Gviz* package is facilitated by so called display parameters. Even though we did not explicitly mention it, we have already made use of them in the previous examples. Display parameters are properties of individual track objects (i.e., of any object inheriting from the base *GdObject* class), and they can be adjusted at various levels:

1. During object instantiation as additional parameters to the constructor function
2. On existing track objects by using the *displayPars()* replacement method
3. Globally for all track objects in a composite plot by supplying additional parameters to *plotTracks()*

The parameter 'stacking' in the *AnnotationTrack* code chunk is an example for 1, and the *add53*, *add35*, and *littleTicks* parameter in the *GenomeAxisTrack* chunk are examples for 3. The following

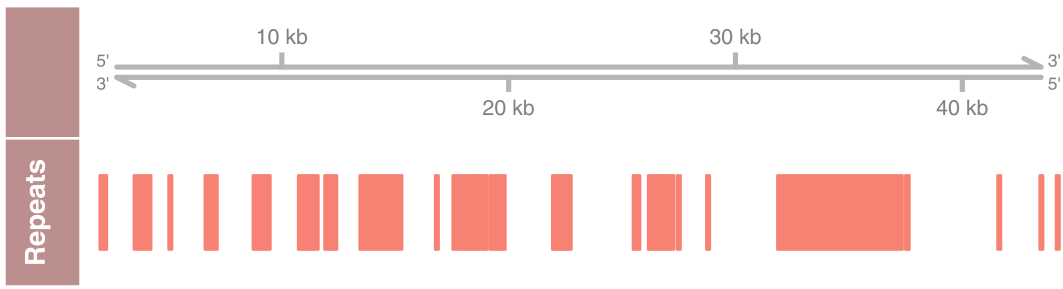

Fig. 4 Annotation track showing the distribution of the first 50 repeat elements on the *Drosophila melanogaster* 3R chromosome with customized plotting colors

code demonstrates the use of the *displayPars()* accessor and replacement methods and produces Fig. 4.

```
head(displayPars(annoTrack))
```

```
## $arrowHeadWidth
## [1] 30
##
## $arrowHeadMaxWidth
## [1] 40
##
## $cex.group
## [1] 0.6
##
## $cex
## [1] 1
##
## $col.line
## [1] "darkgray"
##
## $col
## [1] "darkgray"
```

```
displayPars(annoTrack) <- list(col="salmon", fill=
"salmon")
displayPars(annoTrack, "col")
```

```
## [1] "salmon"
```

```
plotTracks(list(axTrack, annoTrack), background.title=
"rosybrown")
```

In order to make full use of the flexible parameter system we need to know which display parameters control which aspect of a track class. The recommended source for this information are the help pages of the respective track classes, which list all available parameters along with a short description of their effect and their default values in a dedicated 'Display Parameters' section. Alternatively, we can use the *availableDisplayPars()* function in an interactive R session to print out the available parameters for a class as well as their default values in a list-like structure. The single argument to the function is either the name of a track object class, or the object itself, in which case the class is automatically detected.

```
dp <- availableDisplayPars(annoTrack)
tail(dp)
```

```
##
## The following display parameters are available for
'AnnotationTrack' objects:
##(see ? AnnotationTrack for details on their usage)
##
## rotation.item: 0
## rotation.title (inherited from class 'GdObject'): 90
## shape: arrow
## showAxis (inherited from class 'GdObject'): TRUE
## showFeatureId: NULL
## showId: NULL
## showOverplotting: FALSE
## showTitle (inherited from class 'GdObject'): TRUE
## size: 1
## stackHeight (inherited from class 'StackedTrack'): 0.75
## v (inherited from class 'GdObject'): -1
```

To avoid having to change the same parameters over and over again, the *Gviz* package supports display parameter schemes for registering settings in a central location. A scheme is essentially just a couple of nested named lists, where the element names on the first level of nesting correspond to track class names, and the names on the second level to display parameter names. The currently active scheme can be changed by setting the global option *Gviz.scheme*, and a new scheme can be added to the registry by using the *addScheme()* function. There is also a mechanism in place to register schemes upon package loading for even more permanent customization. The *getScheme()* function is useful to get the current scheme as a list structure, for instance to use as a skeleton when creating your own custom scheme. Please note that because display parameters are stored as integral parts of the individual track objects, a scheme change only has an effect on newly created objects.

3.3 A Real World Example

With all the necessary building blocks in place we can now proceed to a somewhat more applied biological example. The data originates from an RNASeq experiment, where the authors were interested in studying the effects of RNAi depletion of a known *Drosophila Melanogaster* splice modulator (*Pasilla*) on alternative splicing of its targets. They were able to identify several genes with effected splicing events, and we want to inspect one of them here in more detail: the *BMM* gene. The most obvious first thing to do is to take a closer look at the *BMM* gene locus, and the *Gviz* package provides the *GeneRegionTrack* class for this purpose. Objects can be created from Bioconductor-specific gene model abstractions (e.g., from *TranscriptDB* objects), or one can retrieve the necessary information from online resources like UCSC or Ensembl Biomart. In this case we select the latter by employing the dedicated *BiomartGeneRegionTrack()* constructor function. All we need to know is the genomic location of our gene of interested. To better put this location in the context of the whole chromosome, we also add our *GenomeAxisTrack* object from before and an *IdeogramTrack* object to the plot. Ideograms are schematic representations of chromosomes, showing their relative size and their cytostain banding patterns. When connected to the Internet, the *Gviz* package will automatically retrieve the necessary chromosome information for a large number of different organisms. The current relative plotting location on a chromosome is always indicated on the ideogram by a red box.

```
chr <- "chr3L"
from <- 14769250
to <- 14779800
geneTrack <- BiomartGeneRegionTrack(genome="dm3",
chromosome=chr, start=from, end=to,
transcriptAnnotation="symbol", name= "Genes")
ideoTrack <- IdeogramTrack(genome="dm3", outline=TRUE)
plotTracks(list(ideoTrack, axTrack, geneTrack), chromosome=chr, from=from,
                      to=to)
```

As we can see in the plot in Fig. 5, there are two transcript variants for the *BMM* gene that differ in the use of a single exon. Now the interesting question is how these exons are differentially used between the wild-type condition and the RNAi knockout of the *Pasilla* gene. The *DEXSeq* package [6] provides some nice infrastructure to make this inference. A full treatment of its features is beyond the scope of this article, and we refer to the package

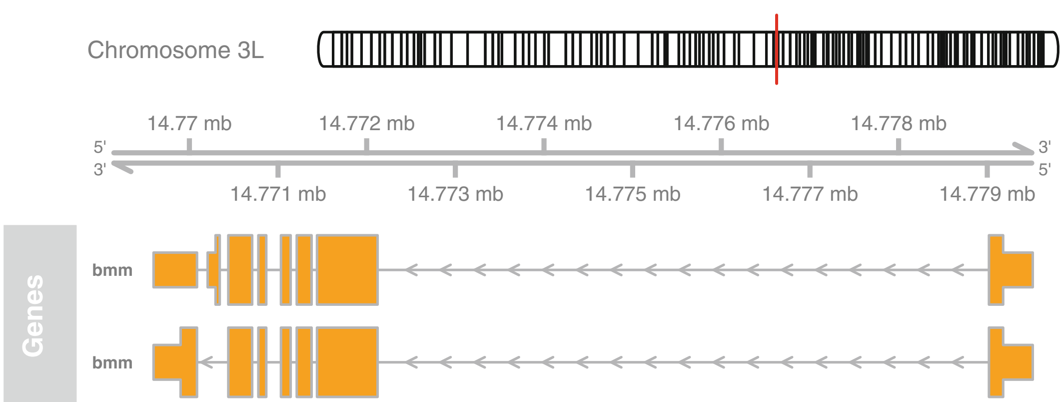

Fig. 5 Ensembl Biomart gene model track showing the Drosophila Melanogaster *BMM* gene locus

vignette for more details. In summary, we load the count data for six samples from the accompanying *pasilla* package, create an object of class *DEXSeqDataSet*, select only a subset of the available genes to speed up the computation and finally fit the differential exon usage coefficients in a generalized linear model.

```
library(pasilla)
library(DEXSeq)

## Loading required package: Biobase
## Welcome to Bioconductor
##
## Vignettes contain introductory material; view with
## 'browseVignettes()'. To cite Bioconductor, see
## 'citation("Biobase")', and for packages 'citation("pkgname")'.
##
## Loading required package: DESeq2
## Loading required package: Rcpp
## Loading required package: RcppArmadillo
## Loading required package: BiocParallel

## converting counts to integer mode
## using supplied model matrix
## using supplied model matrix
```

The resulting *DEXSeqResults* object holds all the information we need to create our visualization, but we have to extract it first in order to build a *DataTrack* object, which is the *Gviz* track class that handles displaying of numeric data. The general idea of the *DataTrack* class is that we can associated one or multiple numbers to a genomic location, representing values from one or several samples. Again, there is a multitude of different inputs for the constructor function, including the use of a *GRanges* object. All its numerical metadata columns will be automatically extracted and used for the track. Because the count data from an RNASeq experiment is often biased by the sequencing library size, we first apply a normalization using the library size factors that have been computed by the *DEXSeq* package. We also want to remove between-exon variation to place the focus on the group differences. Now we can assign these normalized values as metadata columns to the *GRanges* object, and also provide the sample grouping information. From the wide variety of different plotting options we choose a box-and-whisker plot because it nicely contrasts the data distribution between the two groups. We also ask the *Gviz* package to transform the count data to log2 space before plotting. The available horizontal space in this publication is fairly limited, so we will focus on the 3′ part of the *BMM* gene for now.

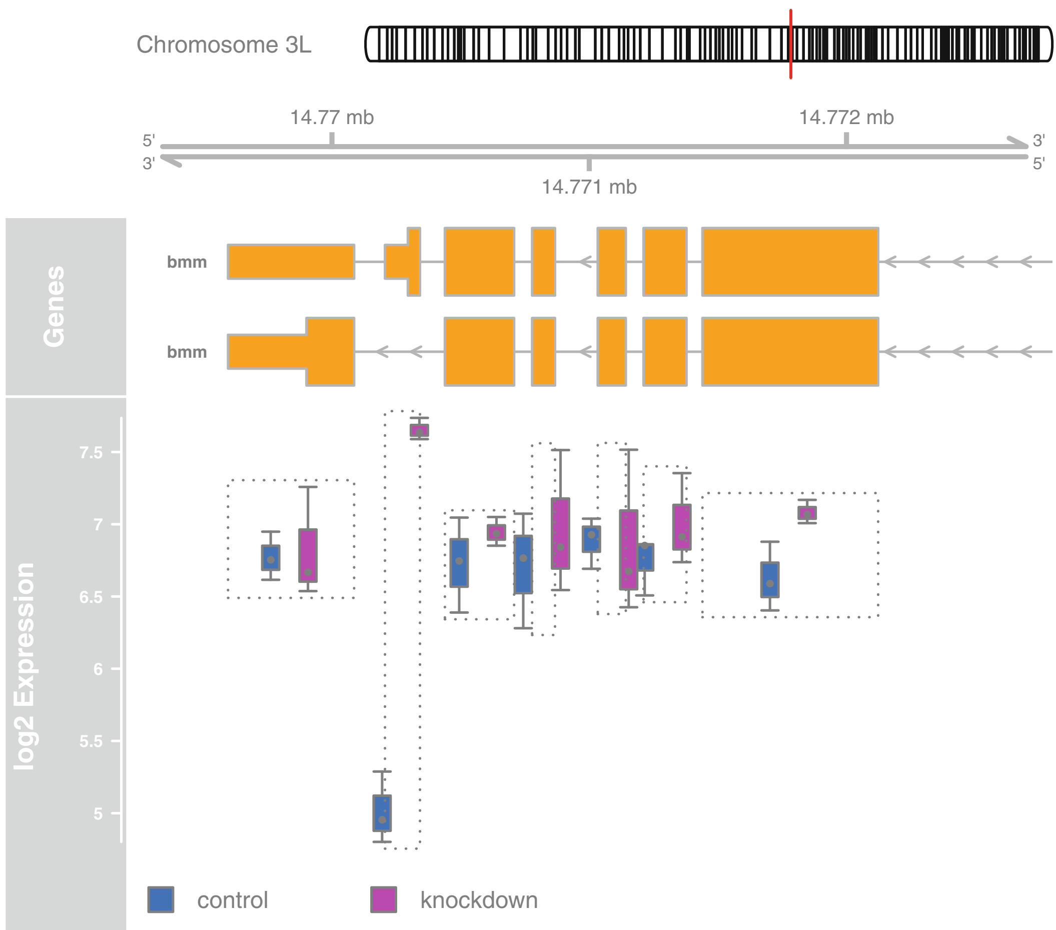

Fig. 6 Ensembl Biomart gene model track and box-and-whisker data track comparing the differential exon usage for the Drosophila Melanogaster *BMM* gene locus as a consequence of the RNAi knockdown of the *Pasilla* splice modulator

```
gr <- granges(dxd)
values(gr) <- (t(t(featureCounts(dxd)) / sizeFactors(dxd))) /
    rowMeans(featureCounts(dxd)) *
mean(featureCounts(dxd))
strand(gr) <- "*"
from2 <- 14769240
to2 <- 14772800
dataTrack <- DataTrack(genome="dm3", range=gr, groups=sampleTable$condition,
                     type="boxplot", box.width=10, size=6,
transformation=function(x) log2(x+1), name="log2 Expression")
plotTracks(list(ideoTrack, axTrack, geneTrack, dataTrack), chromosome=chr,
                   from=from2,
           to=to2, legend=TRUE)
```

The data values for the two groups at each exon in Fig. 6 have been automatically separated, placed at the correct genomic

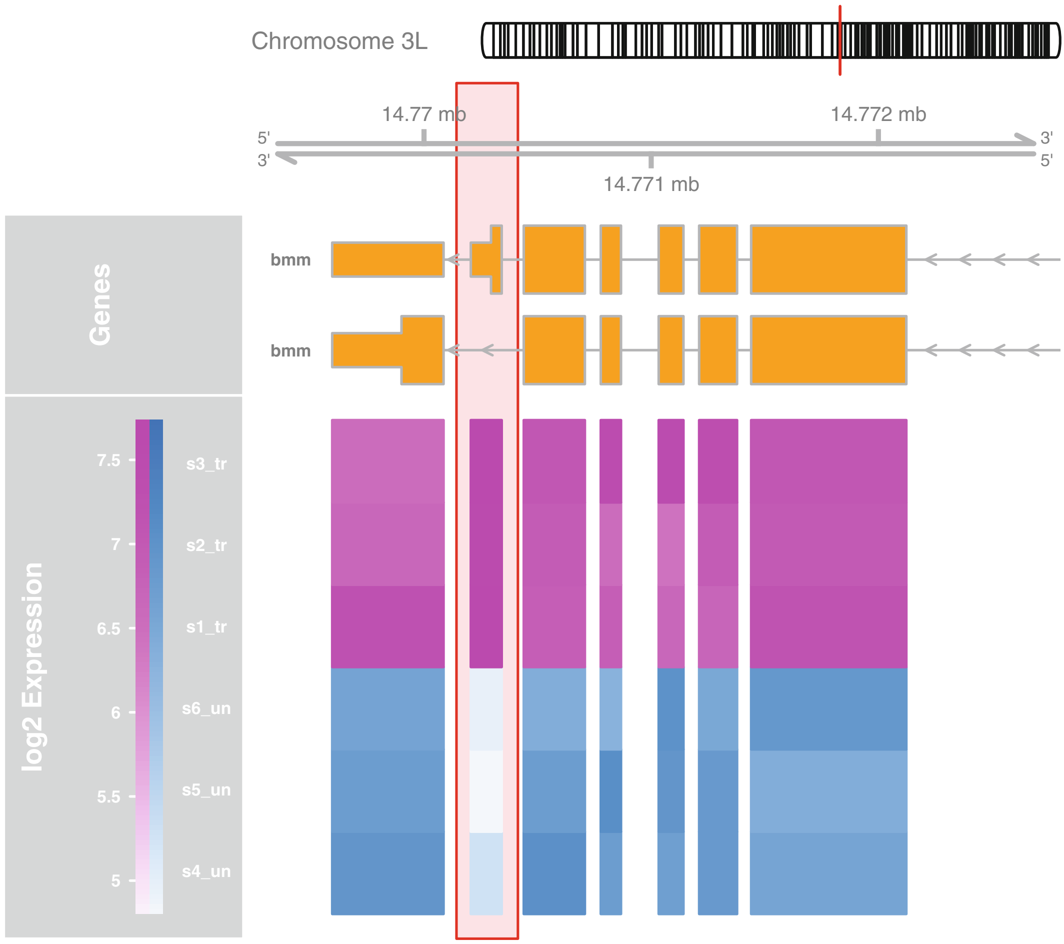

Fig. 7 Ensembl Biomart gene model track and heat map data track comparing the differential exon usage at the Drosophila Melanogaster *BMM* gene locus as a consequence of the RNAi knockdown of the *Pasilla* splice modulator. The location of the most differentially used exon FBgn0036449:2 is highlighted

location and differentially colored for easy contrasting. A vertical scale indicator has been included in the track title panel to show the magnitude of the normalized log expression values. We can immediately see that the second to last exon of the upper *BMM* transcript is used to a much larger extent in the *Pasilla* knockdown samples compared to the wild type. Creating a complex plot like this only took a few lines of code, which really is the fundamental idea behind the *Gviz* package. Another strength is its versatility when it comes to plotting numeric data. For instance we could choose from the long list of available plotting types a heat map instead of the box-and-whisker plot to give the visualization a slightly different spin. One can also draw the attention of the viewer to a particular plot region by wrapping our track objects into a *HighlightTrack* meta object which simply draws a colored box around the provided genomic locations (Fig. 7).

```
plotTracks(HighlightTrack(list(ideoTrack, axTrack, geneTrack, dataTrack),
range=range(ranges(geneTrack)[exon(geneTrack) == "FBgn0036449:2"]) * (-2)),
           chromosome=chr, from=from2, to=to2, type="heatmap", showSample
                    Names=TRUE)
```

The expression values that make up the box plot are actually based on counting the overlaps of a large number of RNAseq read alignments with the individual exons of the *BMM* gene. The alignment raw data for two of the samples is available from the *Gviz* package in the form of two BAM files. If indeed the *FBgn0036449:2* exon is differentially spliced as a consequence of the *Pasilla* knockdown, we should also be able to pick this up in the original read alignments. There should also be even stronger evidence through splice junction spanning reads. i.e., gapped alignments of the same RNA molecule that overlap multiple exons. The *Gviz* package offers functionality for extracting and plotting NGS data directly from BAM files. These files have to be sorted by read coordinates and indexed. Both single end and paired end data are supported. The *AlignmentsTrack* class handles all the details, in particular the streaming of the required data from the BAM file and of course the rendering.

```
bamFiles <- system.file(file.path("extdata", c("GSM461176_S2_DRSC_Untreated_1_
                    bmm.bam",
"GSM461179_S2_DRSC_CG8144_RNAi_1_bmm.bam")), package="Gviz")
readsTrackC <- AlignmentsTrack(range=bamFiles[1], name="Control")
readsTrackK <- AlignmentsTrack(range=bamFiles[2], name="Knockdown")
plotTracks(list(ideoTrack, axTrack, geneTrack, readsTrackC, readsTrackK),
                    chromosome=chr, from=from2, to=to2)
```

As expected, we do see the differential exon usage in the coverage plots in Fig. 8 in the top parts of the read alignments tracks, and also strong evidence for junction spanning reads in the pile-up views in the lower parts, indicated by the connecting lines between the read fragments. Even though it only make limited sense in this context, an *AlignmentsTrack* can also be used to show genotyping details based on the read alignments. It needs to know about the expected reference sequence, though. We can either provide that to the constructor function as a separate argument, or, even better, add a *SequenceTrack* object to our plot which holds an indexed version of the whole *Drosophila Melanogaster* genome. The package will detect the presence of this track and automatically extract the relevant sequence data. A convenient way to create the object is to start from one of the *BSGenome* packages that are provided as part of the Bioconductor suite.

```
library("BSgenome.Dmelanogaster.UCSC.dm3")
## Loading required package: BSgenome
## Loading required package: Biostrings
## Loading required package: rtracklayer
```

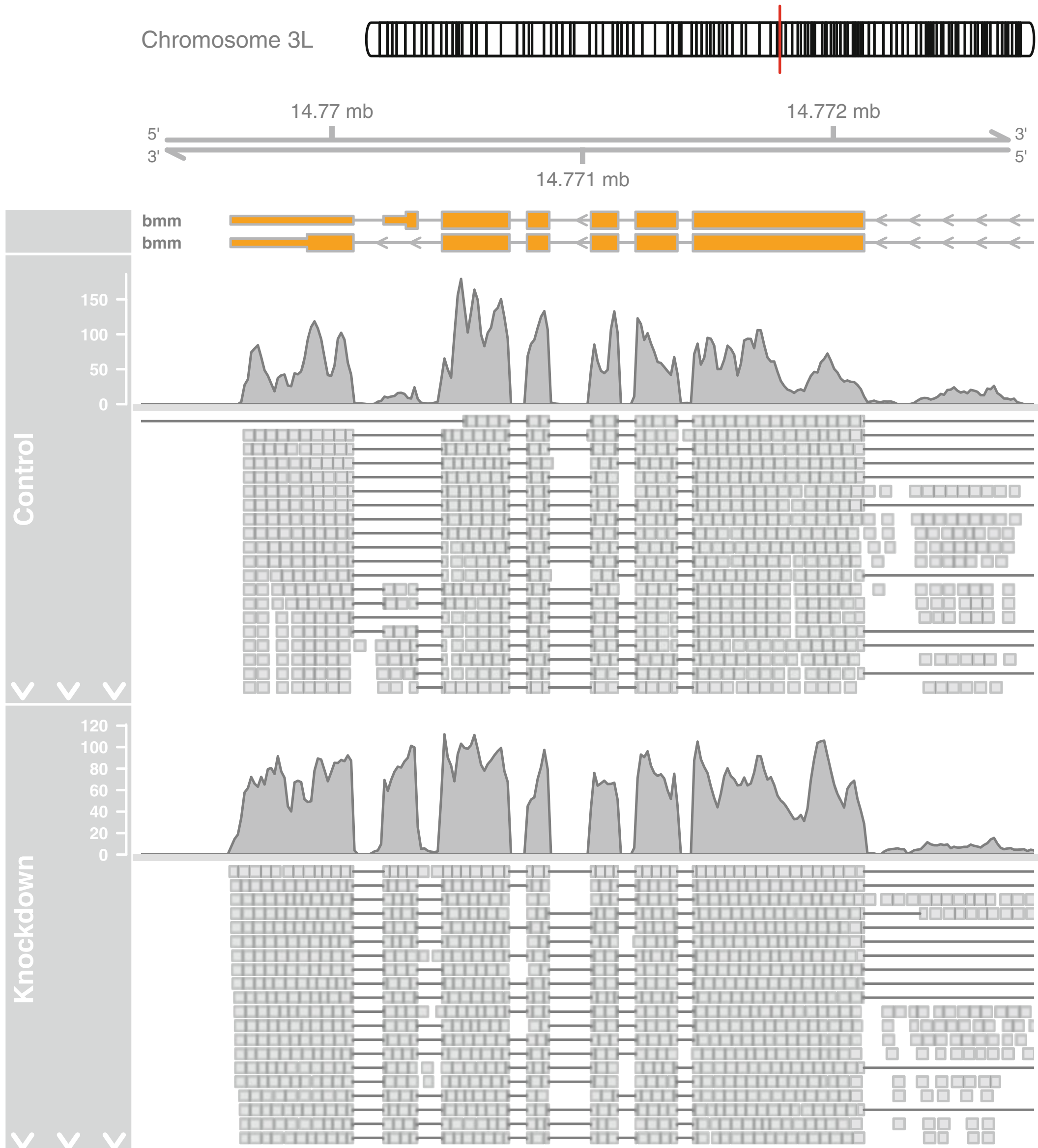

Fig. 8 Aligned RNASeq reads for two samples at the *BMM* gene locus

```
seqTrack <- SequenceTrack(Dmelanogaster)
plotTracks(list(ideoTrack, axTrack, geneTrack, readsTrackK, seqTrack),
                    chromosome= chr, from=from2, to=to2)
```

Mismatches in the read alignments are now highlighted by color-coding, both in the coverage section and in the pile-up section (Fig. 9). When zooming in a little bit we can actually make out locations showing two apparent homozygous SNPs (Fig. 10).

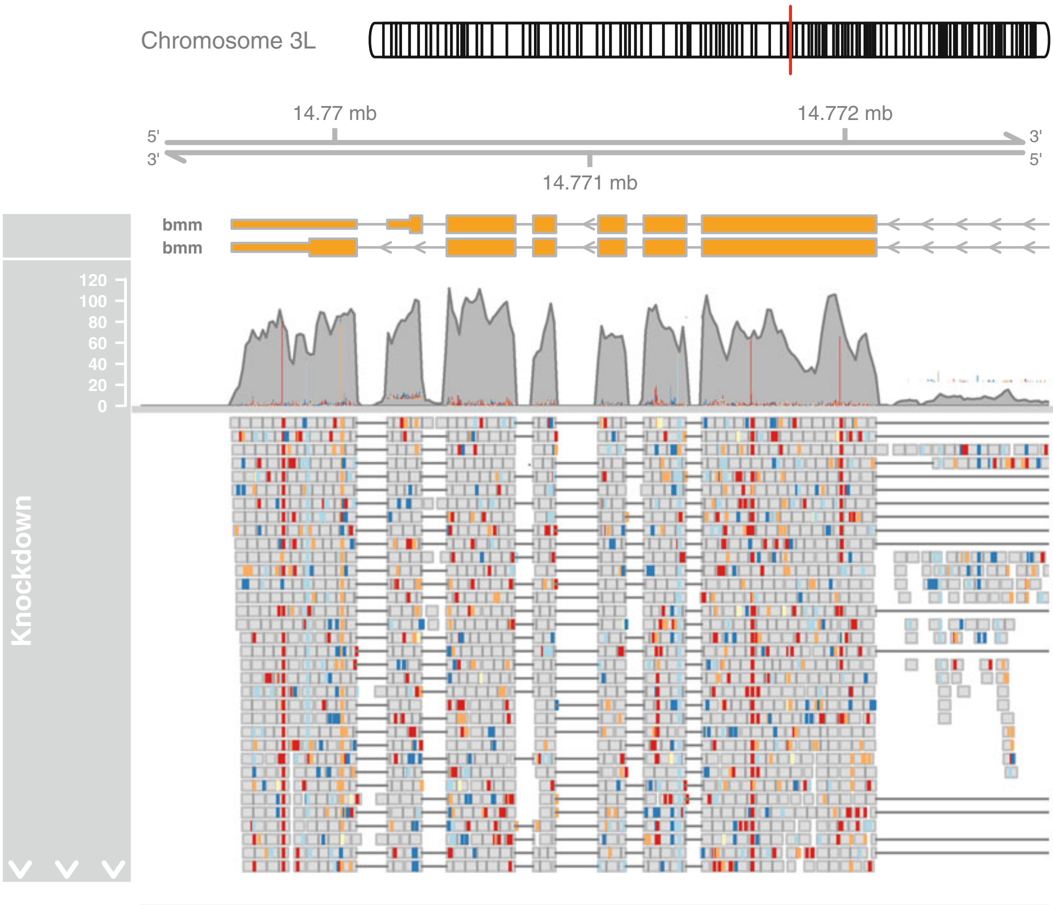

Fig. 9 Aligned RNASeq reads for a single sample at the *BMM* gene locus with alignment mismatch information indicated by color-coding

```
from3 <- 14769770
to3 <- 14769900
plotTracks(list(ideoTrack, axTrack, readsTrackK, seqTrack), chromosome=chr,
                    from=from3, to=to3)
```

Genomic variability is typically much higher in regions that are not highly conserved between closely related species. The UCSC database is a good source for interspecies conservation data, and we can make use of yet another track type in the *Gviz* package to quickly add this information to our plot. To be more precise, we will use the *UcscTrack* meta-constructor to automatically download the relevant data from UCSC and use it for building a *DataTrack* object. The rational here is that one can readily access the underlying UCSC data tables by means of the *rtracklayer* package, and only needs to provide a mapping between the retrieved table columns and their respective *Gviz* track constructor counterparts. An obvious prerequisite to this is that a user needs to know about the

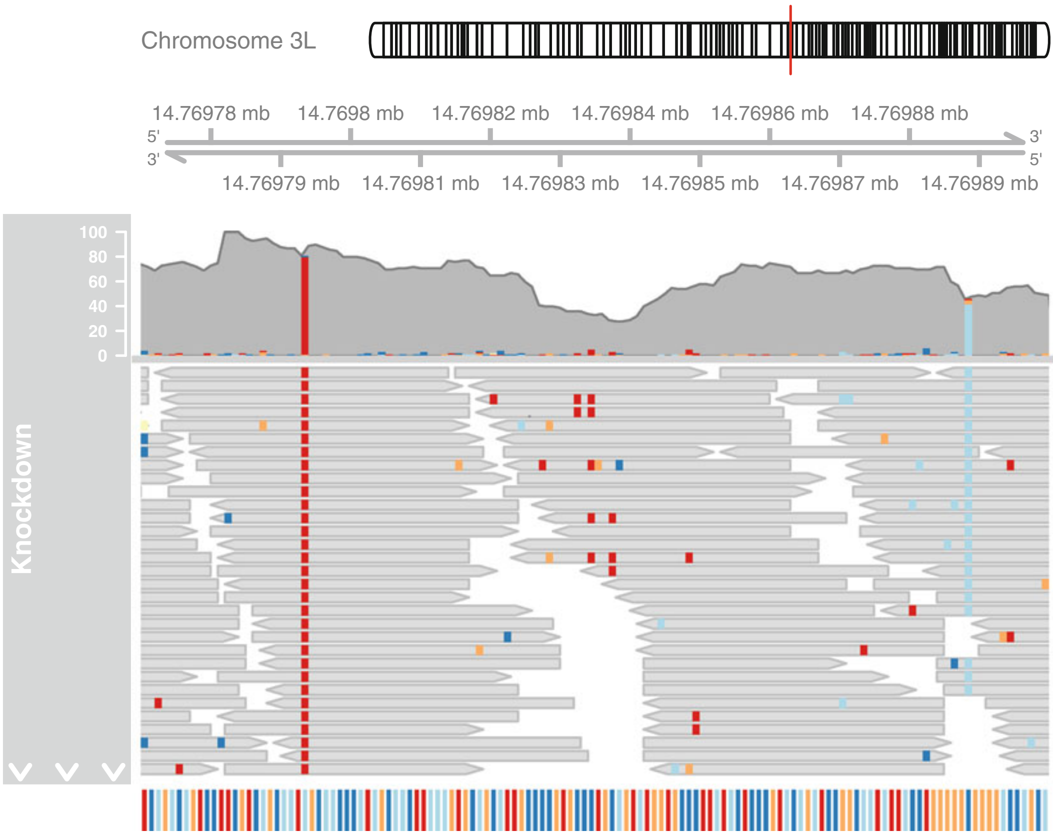

Fig. 10 Zoomed in view on aligned RNASeq reads for a single sample with alignment mismatch information indicated by color-coding showing two homozygous SNPs

underlying structure of the UCSC data table, which can easily be inspected via the online UCSC table browser. In the following code chunk we apply exactly this approach to the *phastCons15way* table within the UCSC *Conservation* track. The *trackType* argument tells the meta-constructor to build a *DataTrack* object, and the *start*, *end* and *data* arguments provide the mapping from the table columns to the *DataTrack* constructor arguments. We also add some smoothing by applying a running window summarization and slightly adjust the colors and the relative track size.

```
consTrack <- UcscTrack(genome="dm3", chromosome=chr, track="Conservation",
                    table="phastCons15way",
from=from3,  to=to3,  trackType="DataTrack",  start="start",  end="end",
                    data="score", type="histogram",
window="auto", col.histogram="darkgray", fill.histogram="darkgray", name="%
                    Cons", size=6)
plotTracks(list(ideoTrack, axTrack, readsTrackK, seqTrack, consTrack), chro-
                    mosome=chr, from=from3, to=to3)
```

As seen in Fig. 11, both SNPs do fall in relatively unconserved regions.

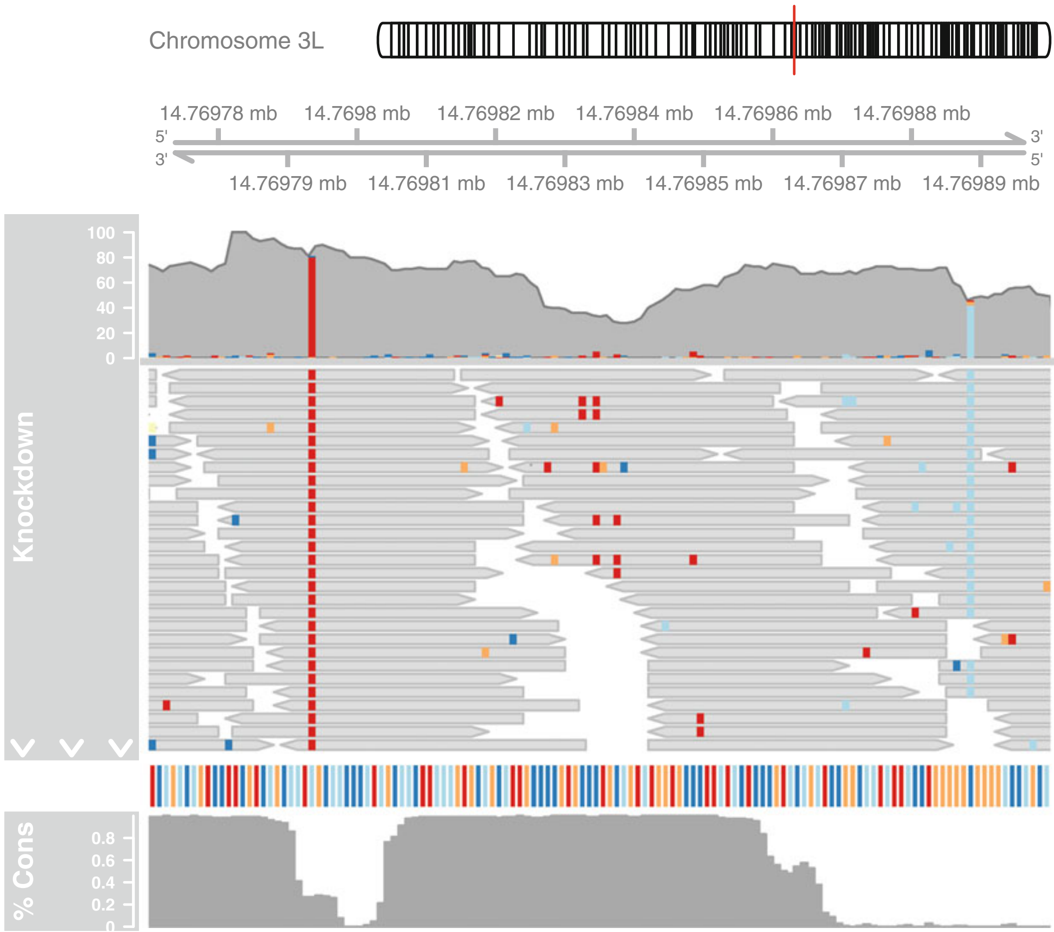

Fig. 11 Zoomed in view on aligned RNASeq reads for a single sample with alignment mismatch information indicated by color coding showing two homozygous SNPs. The interspecies conservation of the locus is shown in the additional data track

4 Notes

In this chapter, we described how to visualize different features along genomic coordinates using the *Gviz* package. Similar to other Bioconductor packages for genomic data visualization, *Gviz* is using the concept of aligned tracks representing different annotation and numerical data types which are combined to produce the final figure for a given genome region. *Gviz* can load data from both commonly used genome browsers (Ensembl Genome Browser [7], UCSC Genome Browser [8]) and various file formats, and offers a large collection of plotting types for numerical data. The package empowers users to generate figures almost identical to or often times better than the seemingly omnipresent screenshots from genome browsers in a much more flexible way. It even offers zoom-in capability to single-base-pair resolution similar to the Integrated Genome Viewer [9], which is often useful to highlight sequence variation identified in the analyzed samples. Compared to more "point and click-like" approaches by other tools, *Gviz* does not lend itself very much to interactive data browsing, even though there are smart caching and streaming mechanisms in place to speed up the plotting. That said, excellent new packages like *shiny* [10] and *rmarkdown* [11] provide some opportunity to create interactive documents for data exploration using *Gviz* plots. More importantly, though, in combination with tools like *Sweave* [12] or *knitr* [13], *Gviz* can be used to generate high quality figures in a scripted fashion, thus facilitating reproducible research. Its tight integration in R and Bioconductor environment allows users to perform the entire analysis from the very start to the final figure generation within a single software tool.

References

1. Lawrence M, Gentleman R, Carey V (2009) Rtracklayer: An r package for interfacing with genome browsers. Bioinformatics 25:1841–1842. doi:10.1093/bioinformatics/btp328
2. Durinck S, Bullard J. GenomeGraphs: plotting genomic information from ensembl. R package version 1.30.0
3. Brooks AN, Yang L, Duff MO, Hansen KD, Park JW, Dudoit S, Brenner SE, Graveley BR (2011) Conservation of an rNA regulatory map between drosophila and mammals. Genome Res 21:193–202. doi:10.1101/gr.108662.110
4. Trapnell C, Pachter L, Salzberg SL (2009) TopHat: discovering splice junctions with rNA-seq. Bioinformatics 25:1105–1111. doi:10.1093/bioinformatics/btp120
5. Ensembl (2011) Drosophila melanogaster genome version BDGP5.25.62
6. Anders S, Reyes A, Huber W (2012) Detecting differential usage of exons from rNA-seq data. Genome Res 22:2008–2017. doi:10.1101/gr.133744.111
7. Cunningham F, Amode MR, Barrell D, Beal K, Billis K, Brent S, Carvalho-Silva D, Clapham P, Coates G, Fitzgerald S, Gil L, Girón CG, Gordon L, Hourlier T, Hunt SE, Janacek SH, Johnson N, Juettemann T, Kähäri AK, Keenan S, Martin FJ, Maurel T, McLaren W, Murphy DN, Nag R, Overduin B, Parker A, Patricio M, Perry E, Pignatelli M, Riat HS, Sheppard D, Taylor K, Thormann A, Vullo A, Wilder SP, Zadissa A, Aken BL, Birney E, Harrow J, Kinsella R, Muffato M, Ruffier M, Searle SM, Spudich G, Trevanion SJ, Yates A, Zerbino DR, Flicek P (2015) Ensembl 2015. Nucleic Acids Res 43: D662–D669. doi:10.1093/nar/gku1010
8. Kent WJ, Sugnet CW, Furey TS, Roskin KM, Pringle TH, Zahler AM, Haussler D (2002) The human genome browser at uCSC. Genome Res 12:996–1006. doi:10.1101/gr.229102
9. Robinson JT, Thorvaldsdottir H, Winckler W, Guttman M, Lander ES, Getz G, Mesirov JP (2011) Integrative genomics viewer. Nat Biotech 29:24–26
10. Chang W (2015) Shiny: web application framework for r
11. Allaire J, Cheng J, Xie Y, McPherson J, Chang W, Allen J, Wickham H, Hyndman R (2015) Rmarkdown: dynamic documents for r
12. Leisch F (2002) Sweave: dynamic generation of statistical reports using literate data analysis. In: Härdle W, Rönz B (eds) Compstat. Physica-Verlag HD, Berlin, pp 575–580
13. Xie Y (2014) Knitr: a comprehensive tool for reproducible research in R. In: Stodden V, Leisch F, Peng RD (eds) Implementing reproducible computational research. Chapman and Hall/CRC, Boca Raton. ISBN 978-1466561595