# *TarSeqQC*: Targeted Sequencing Experiment Quality Control

Gabriela A Merino[1], Cristóbal Fresno[1], Yanina Murua[2], Andrea S Llera[2], and Elmer A Fernández[1]

[1]CONICET-Universidad Católica de Córdoba, Argentina
[2]Laboratory of Molecular and Cellular Therapy, Instituto Leloir-CONICET, Argentina

October 29, 2019

gmerino@bdmg.com.ar

**Abstract**

Targeted Sequencing experiments are a Next Generation Sequencing application, designed to explore a small group of specific genomic regions. The *TarSeqQC* package models this kind of experiments in R, and its main goal is to allow the quality control and fast exploration of the experiment results. To do this, a new R class, called *TargetExperiment*, was implemented. This class is based on the *Bed File*, that characterize the experiment, the alignment *BAM File* and the reference genome *FASTA File*. When the constructor is called, coverage and read count information are computed for the targeted sequences. After that, exploration and quality control could be carried out using graphical and numerical tools. Density, bar, read profile and box plots were implemented to achieve this task. A circular histogram plot was also implemented in order to summarize all experiment results. Coverage or median count intervals can be defined and explored to further assist quality control analysis. Library and pool preparation, sequencing errors, fragment length or GC content bias could be easily detected. Finally, an .xlsx file reporting quality control results can be built.
Since its 1.1.8 version, *TarSeqQC* implements a new class, *TargetExperimentList* in order to allow the comparison and joint analysis of several Targeted Sequencing experiments performed using the same *Bed File*. This class includes several plots that can help to analyze several samples from the same patient or from different patients under the same pathological condition.

# Contents

# 1 Introduction

Next Generation Sequencing (NGS) technologies produce a huge volume of sequence data at relatively low cost. Among the different NGS applications, Targeted Sequencing (TS) allows the exploration of specific genomic regions, called *features*, of a small group of genes (Metzker, 2010). An ordinary application of TS is to detect Single Nucleotide Polymorphisms (SNPs) involved in several pathologies. Nowadays, *TS cancer panels* are emerging as a new screening methodology to explore specific regions of a small number of genes known to be related to cancer. In TS, specific regions of a DNA sample are copied and amplified by PCR. If a target region is too large, several primers can be used to read it. In addition, if the panel also has a large number of interest genomic regions, different PCR pools could be required in order to achieve a good coverage. All DNA fragments are sequenced in an NGS machine, generating millions of short sequence reads, though less than if the whole genome was sequenced. Reads are then aligned against a reference genome and, after that, a downstream analysis could be performed. However, prior to this, it is crucial to evaluate the run performance, as well as the experiment quality control, i.e., how well the features were sequenced, which feature and gene coverages were achieved, if some problems arise in the global setting or by specific PCR pools (Metzker, 2010).

At present, several open access tools can be used to explore and control experiment results(Lee et al., 2012). Those tools allow visualization and some level of read profiles quantification. But, they were developed as general purpose tools to cover a wide range of NGS applications, mainly for whole genome exploration. Consequently, they require a great amount of computational resources and power. On the other hand, in TS only small group of regions, the features, are required to be explored and characterized in terms of coverage or counts, as well as, the evaluation and comparison of pool efficiency. In addition, this analysis should be also performed at a gene level in order to provide a general results overview. In this scenario, current genomic tools have become heavy and coarse for such amount of data. Consequently, the availability of light, fast and specific tools for TS data handling and visualization is a must in current labs.

Here we present *TarSeqQC* R package, an exploration tool for fast visualization and quality control of TS experiments. Its use is not restricted to TS and can also be used to analyze data from others NGS applications in which *feature-gene* structure could be defined, like exons or isoforms in RNA-seq and amplicons in DNA-seq.

This vignette intends to guide through to the use of the *TarSeqQC* R Bioconductor package. First, the input data format is described. Then, we show how to build an instance of the *TargetExperiment* class. After that, we will graphically explore the results and do the quality control over the sequenced features. Finally, we will build an .xlsx report that summarize the analysis above.

# 2 Preliminaries

## 2.1 Installation

*TarSeqQC* is a package for the R computing environment and it is assumed that you have already installed R. See the R project at http://www.r-project.org. To install the latest version of *TarSeqQC*, you will need to be using the latest version of R. *TarSeqQC* is part of the Bioconductor project at http://www.bioconductor.org. To get the *TarSeqQC* package you can use:

```
> if (!requireNamespace("BiocManager", quietly=TRUE))
+      install.packages("BiocManager")
> BiocManager::install("TarSeqQC")
```

Like other Bioconductor packages, there are always two available versions of *TarSeqQC*. Most users will use the current official release version, which will be installed by 'BiocManager::install' if you are using the current version of R. There is also a developmental version of the package that includes new features due for the next official Bioconductor release. The developmental version will be installed if you are using the developmental version of R. Both can be found in the Bioconductor site.

## 2.2 Citation

The *TarSeqQC* R package can be cited using:

```
> citation("TarSeqQC")
```

```
  Merino GA, Murua YA, Fresno C, Sendoya JM, Golubicki M, Iseas S, Coraglio M,
  Podhajcer OL, Llera AS and Fernandez EA (2017): TarSeqQC: Quality control on
  targeted sequencing experiments in R. Human Mutation 38 (5), 494-502

A BibTeX entry for LaTeX users is

  @Article{,
    title = {TarSeqQC: Quality control on targeted sequencing experiments in
        R},
    author = {Gabriela A. Merino and Yanina A. Murua and Cristobal Fresno and Juan M. Sendoya and Mariano
    year = {2017},
    journal = {Human Mutation},
    volume = {38},
    issue = {6},
    pages = {494-502},
    doi = {10.1002/humu.23204},
    url = {http://onlinelibrary.wiley.com/doi/10.1002/humu.23204/abstract},
  }
```

# 3 The *TargetExperiment* class

*TarSeqQC* R package is based on the *TargetExperiment* class. The Figure 1 shows the *TargetExperiment* class structure.

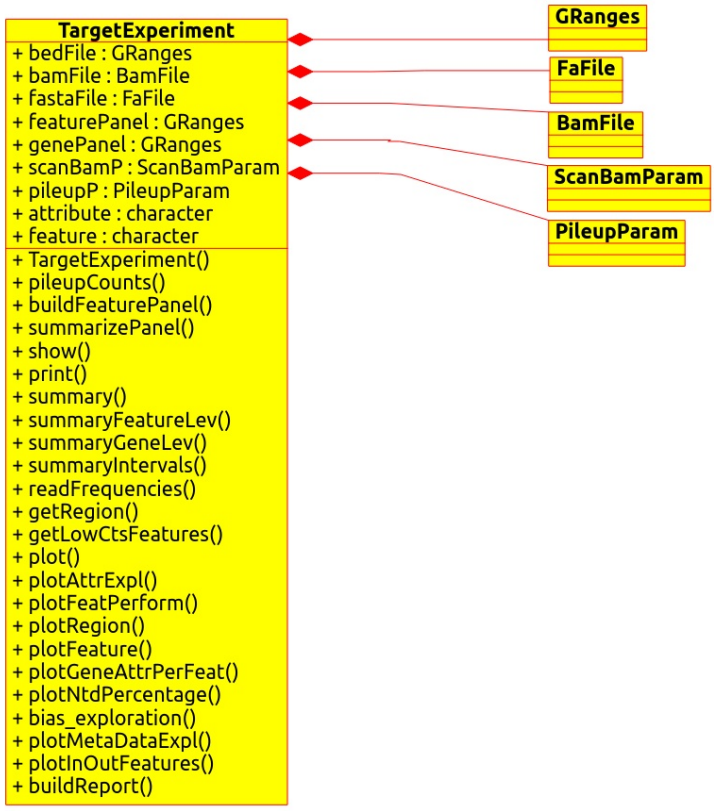


Figure 1: *TargetExperiment* class diagram.

The *TargetExperiment* class has nine slots:

- `bedFile`: a *GRanges* object that models the *Bed File*
- `bamFile`: a *BamFile* object that is a reference to the *BAM File*.
- `fastaFile`: a *FaFile* object that is a reference to the reference sequence file.
- `featurePanel`: a *GRanges* object that models the feature panel and related statistics.
- `genePanel`: a *GRanges* object that models the analyzed panel and related statistics at a gene level.
- `scanBamP`: a *ScanBamParam* containing the information to scan the *BAM File*.
- `pileupP`: a *PileupParam* containing the information to build the pileup matrix.

- `attribute`: a *character* indicating which attribute *coverage* or *medianCounts* will be used to the analysis.
- `feature`: a *character* indicating the name of the analyzed features, e.g.: "amplicon", "exon", "transcript".

The next sections will illustrate how the *TargetExperiment* methods can be used. For this, the *TarSeqQC* R package provides a *Bed File*, a *BAM File*, a *FASTA File* and a dataset that stores the *TargetExperiment* object built with those. This example case is based on a synthetic amplicon sequencing experiment containing 29 *amplicons* of 8 genes in 4 chromosomes.

## 3.1 Input Data

A TS experiment is characterized by the presence of a *Bed File* which defines the *features* that should be sequenced. The *TarSeqQC* package follows this architecture, where the *Bed File* is the key data of the experiment. However, *TarSeqQC* also requires mainly three pieces of information that should be provided in order to call the *TargetExperiment* constructor. The *Bed File*, the *BAM File*, that contains the obtained alignment for the sequenced reads, and the sequence *FASTA File*. The complete path to these files should be defined when the *TargetExperiment* constructor is called.
Other parameters can also be specified in the *TargetExperiment* object constructor. The `scanBamP` and `pileupP` are instances of the *ScanBamParam* and *PileupParam* classes defined in the *RSamtools* R Bioconductor package (Morgan et al., 2015b). These parameters specify how to scan the *BAM File* and how to build the corresponding *pileup*, that will be used for exploration and quality control. The `scanBamP` allows the specification of the features specified in the *Bed File*, according to Morgan et al. (2015b) specifications. The `pileupP` establishes what information should be contained in the pileup matrix, for instance, if nucleotides and/or strand should be distinguished. If these two parameters are not specified, the default values of their constructors will be used.
Other important parameters that should be specified before to conduct the *Quality Control* are `feature` and `attribute`. The first is a character that determines which kind of features are contained in the *Bed File*. In the example presented here, *amplicon* is the feature type. The second parameter, `attribute`, can be *coverage* or *medianCounts* defining which measures will be considered in the *Quality Control* analysis.

### 3.1.1 Bed File

The *Bed File* is stored as a *TargetExperiment* slot and it is modeled as a *GRanges* object (Lawrence et al. (2013)). The *Bed File* must be a tabular file in which each row represents one feature. This file should contain at least "chr", "start", "end" , "name" and "gene" named columns. Additional columns like "strand" or another experimental information could be included and would be conserved. For example, in some experiments, more than one PCR pool is necessary. In this case, the *Bed File* must also contain a "pool" column specifying the pool in which each feature was defined. This information is an imperative requisite to evaluate the performance of each PCR pool.
A *GRanges* object represents a collection of genomic features each having a single start and end location on the genome (Lawrence et al., 2013). In order to use it to represent the *Bed File*, the "chr", "start" and "end" mandatory fields will be used to define the "seqnames", "start" and "end" *GRanges* slots. The same will occur if the optional field "strand" is included in the *Bed File*. The "name" column will be set to ranges identifiers. Finally, "gene" and additional columns like "pool", will be stored as metadata columns.
In order to create a *TargetExperiment* object, the complete route to the *Bed File* and its name must be specified as a *character* R object. Thus, to use the example *Bed File* provided by *TarSeqQC*:

```
> bedFile<-system.file("extdata", "mybed.bed", package="TarSeqQC", mustWork=TRUE)
```

Note that any experiment, in which can be defined ***feature-gene*** relations, could be analyzed using the *TarSeqQC* R Bioconductor package. For instance, if you have an RNA-seq experiment and you are interested in exploring some genes, you could build your customized *Bed File* in which the *feature* could be "exon" or "transcript".

### 3.1.2 BAM File

The *BAM File* stores the ordered alignment results (Li et al., 2009). In this example case, it corresponds to the amplicon sequencing experiment alignment. This file will be used to build the *pileup* matrix for the selected features. Briefly, a *pileup* is a matrix in which each row represents a genomic position and have at least three columns: "pos", "chr" and "counts". The first and second columns specify the genomic position and "counts" contains the total read counts for this position. Pileup matrix could contain four additional columns that store the read counts for each nucleotide at this position.
In order to call the *TargetExperiment* constructor, the complete route to the *BAM File* and its name must be specified as a *character* R object. For example, we can define it in order to use *TargetExperiment* external data:

```
> bamFile<-system.file("extdata", "mybam.bam", package="TarSeqQC", mustWork=TRUE)
```

When the *TargetExperiment* constructor is called the *BAM File*, will be stored as a *BamFile* object (Morgan et al., 2015b) and this object will be a *TargetExperiment* slot.

### 3.1.3 FASTA File

The *FASTA File* contains the reference sequence previously used to align the *BAM File* and will be used to extract the feature sequences. This information is useful to compare the pileup results with the reference, in order to detect potential *nucleotide variants*. To create a *TargetExperiment* object, the full path to the *FASTA File* and its name must be specified as a character R object. For example:

```
> fastaFile<-system.file("extdata", "myfasta.fa", package="TarSeqQC",
+                        mustWork=TRUE)
```

The *FASTA File* will be stored as a *FaFile* object (Morgan et al. (2015b)) and this object will be set to a *TargetExperiment* slot.

### 3.1.4 Additional input data

The previous files are mandatory to call the *TargetExperiment* constructor. Additional parameters can be set in order to apply several methods and perform quality control and results exploration. These parameters are:

- `scanBamP`: is a *ScanBamParam* object, that specifies rules to scan a *BamFile* object. For example, if you wish only keep those reads that were properly paired, or those that have a specific Cigar code,

`scanBamP` can be used to specify it. In TS experiments, we want to analyze only the features. The way to specify this is using the `which` parameter in the *scanBamP* constructor. If the `scanBamP` parameter was not specified in the *TargetExperiment* constructor calling, its default value will be used and then, the `which` parameter will be specified using the *Bed File*.

- `pileupP`: is a *PileupParam* object, that specifies rules to build the *pileup* starting from a *BamFile*. You can use the `pileupP` parameter to specify if you want to distinguish between nucleotides and or strands, filter low read quality or low mapping quality bases. If the `pileupP` parameter is not specified, its default value will be used.
- `attribute`: is a *character* that specifies which attribute must be used for the results exploration and quality control. The user can choice between *medianCounts* or *coverage*. If the *attribute* parameter is not specified in the *TargetExperiment* constructor, it will be set to `""`. But, prior to performing some exploration or control, this argument must be set using the `setAttribute()` method.
- `feature`: is a *character* that defines what means a *feature*. In this vignette a little example using a synthetic amplicon targeted sequencing experiment is shown, thus the feature means an *amplicon*. But, the use of *TarSeqQC* R package is not restricted to analyze only this kind of experiments. If you don't specify the `feature` parameter, it will be set to `""`. But, as in the `attribute` parameter, it must be set prior to performing some exploration or control. It can be done using the `setFeature()` method.
- `BPPARAM`: is a *BiocParallelParam* instance defining the parallel back-end to be used during evaluation (see (Morgan et al., 2015a)). It allows the specification of how many `workers` (CPUs) will be used, etc.

For more information about *ScanBamParam* and *PileupParam* constructors see *Rsamtools* manual.

## 3.2 Creating a *TargetExperiment* object

### 3.2.1 Controlling files consistency

*TarSeqQC* provides a specific method that allows the inspection of the *Bed File* and *FASTA File* consistency previous to the *TargetExperiment* constructor call. The `checkBedFasta` method receives the full path to those files and as it is executed it is printing several messages into the screen. Duplicated features location or names, incorrect start/end positions and inconsistency between the features and the reference genome will be analyzed.

```
> checkBedFasta(bedFile , fastaFile)

- Checking bed file, mandatory columns:
 OK
- Checking bed file, duplicated feature IDs:
 OK
- Checking bed file, duplicated feature locations:
 OK
- Checking bed file, start and end values:
 OK
- Checking bed-fasta chromosome consistency:
 OK
- Checking bed-fasta genomic coordinates consistency:
 OK
```

### 3.2.2 Constructor call

Once you have defined the input data presented above, the *TargetExperiment* constructor could be called using:

```
> library("TarSeqQC")
> library("BiocParallel")
> BPPARAM<-bpparam()
> myPanel<-TargetExperiment(bedFile, bamFile, fastaFile, feature="amplicon",
+                           attribute="coverage", BPPARAM=BPPARAM)
```

When `TargetExperiment` is called, some *TargetExperiment* methods are invoked in order to define two of the *TargetExperiment* slots. First, the `buildFeaturePanel` is internally used in order to build the `featurePanel` slot. Then, the `summarizePanel` is invoked in order to build the `genePanel` slot, which contains the information summarized at a gene level.
In the previous example, the `feature` and `attribute` parameter values were defined. If they aren't specified in the constructor call, the *TargetExperiment* object can be created, but a warning message will be printed. After that, the `setFeature` and `setAttribute` methods should be used to set these values. For example:

```
> # set feature slot value
> setFeature(myPanel)<-"amplicon"
> # set attribute slot value
> setAttribute(myPanel)<-"coverage"
```

As mentioned earlier, when the `scanBamP` and `pileupP` are not specified in the constructor call, they assume their default constructor. But, after the constructor call, they could be specified using `setScanBamP` and `setPileupP` methods.

```
> # set scanBamP slot value
> scanBamP<-ScanBamParam()
> #set scanBamP which slot
> bamWhich(scanBamP)<-getBedFile(myPanel)
> setScanBamP(myPanel)<-scanBamP
> # set pileupP slot value
> setPileupP(myPanel)<-PileupParam(max_depth=1000)
> # build the featurePanel again
> setFeaturePanel(myPanel)<-buildFeaturePanel(myPanel, BPPARAM)
> # build the genePanel again
> setGenePanel(myPanel)<-summarizePanel(myPanel, BPPARAM)
```

Note that the previous code specifies that the maximum read depth can be 1000. If you have some genomic positions that have more than 1000 reads, they will be truncated to this number. It is also important to note that, if any change in the `scanBamP` and/or `pileupP` slots was done, the `featurePanel` and the `genePanel` slots will be set again.
The *TarSeqQC* R package provides a dataset that stores the *TargetExperiment* object built in the previous steps. To use it, run:

```
> data(ampliPanel, package="TarSeqQC")
```

The loaded object is called *ampliPanel*. As was mentioned before, some *TargetExperiment*() methods need to consult the *BAM File* and *FASTA File*, and for this, the `bamFile` and `fastaFile` slots are used. Given that *ampliPanel* was built by the package creators, the path file routes that have its slots are not the same in which these files are located on users' computers. Thus, after *ampliPanel* loading, it is necessary to re-define the *BAM File* and *FASTA File* path files, running:

```
> # Defining bam file and fasta file names and paths
> setBamFile(ampliPanel)<-system.file("extdata", "mybam.bam",
+      package="TarSeqQC", mustWork=TRUE)
> setFastaFile(ampliPanel)<-system.file("extdata", "myfasta.fa",
+      package="TarSeqQC", mustWork=TRUE)
```

Note that `featurePanel` and `genePanel` do not need to be rebuilt. The redefinition of the file names is only necessary in order to use *TargetExperiment* methods that query this files.

### 3.2.3 Early exploration

The *TargetExperiment* class has typical `show/print` and `summary` R methods implemented. In addition, the `summaryGeneLev` and `summaryFeatureLev` methods allow the summary exploration at "gene" and "feature" level. The next example illustrates how these methods can be called:

```
> # show/print
> myPanel
```

```
TargetExperiment
amplicon panel:
        GRanges object with 3 ranges and 6 metadata columns:
        seqnames    ranges strand |        gene        gc  coverage sdCoverage
           <Rle> <IRanges>  <Rle> | <character> <numeric> <numeric>  <numeric>
  AMPL1     chr1   463-551      * |       gene1     0.674       320         19
  AMPL2     chr1 1553-1603      * |       gene2     0.451       550         90
  AMPL3     chr1 3766-3814      * |       gene2     0.531       455         12
        medianCounts IQRCounts
           <numeric> <numeric>
  AMPL1          326        24
  AMPL2          574        14
  AMPL3          463        27
  -------
  seqinfo: 4 sequences from an unspecified genome; no seqlengths

gene panel:
        GRanges object with 3 ranges and 4 metadata columns:
        seqnames    ranges strand | medianCounts IQRCounts  coverage sdCoverage
           <Rle> <IRanges>  <Rle> |    <numeric> <numeric> <numeric>  <numeric>
  gene1     chr1   463-551      * |          326         0       320          0
```

```
  gene2     chr1 1553-3814      * |          518        56       502         67
  gene3     chr3      1-59      * |            0         0         0          0
  -------
  seqinfo: 4 sequences from an unspecified genome; no seqlengths

selected attribute:
         coverage


> # summary
> summary(myPanel)


         Min. 1st Qu. Median Mean 3rd Qu. Max.
gene        0     261    328  312     396  502
amplicon    0     143    288  316     472  931


> #summary at feature level
> summaryFeatureLev(myPanel)


         Min. 1st Qu. Median Mean 3rd Qu. Max.
amplicon    0     143    288  316     472  931


> #summary at gene level
> summaryGeneLev(myPanel)


     Min. 1st Qu. Median Mean 3rd Qu. Max.
gene    0     261    328  312     396  502
```

Using those methods you can easily find, for example, that amplicons were sequenced,in average, at a coverage of 312. It can also be observed that there is at least one amplicon that was not read. This is because the minimum value of the attribute (*coverage*) is 0. To complement this analysis, the attribute distribution can be explored using:

```
> g<-plotAttrExpl(myPanel,level="feature",join=TRUE, log=FALSE, color="blue")
> x11(type="cairo");
> g
```

In Figure 2, the `join` parameter was set to `TRUE`. Thus, density and box plots are plotted together. If it is set to `FALSE`, the figure will contain the attribute box-plot on the left and the corresponding attribute density plot on the right.
Exploration of any metadata information is also provided. *GC content, feature length* distributions and *gene* or *pool* frequencies can be explored using the `plotMetaDataExpl` method. Other metadata columns, specified in the *bedFile*, can also be analyzed. If the analyzed metadata is numeric, then boxplot and density plot is built. Those can be plotted together, or not, and in log10 scale. On the other hand, if it is categorical like *gene* or *pool*, a bar frequency (absolute or in relative percentages) is plotted. The following code allows the exploration of feature length distributions and gene frequencies along the features.

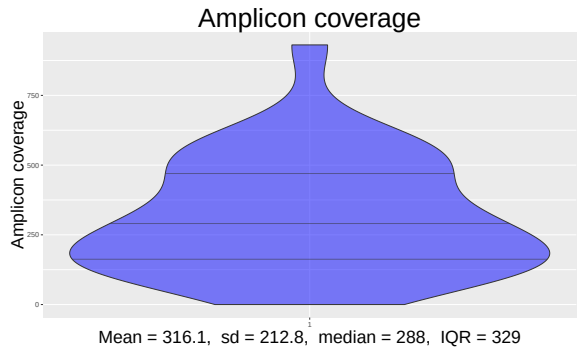


Figure 2: Attribute distribution and density plots.

```
> # explore amplicon length distribution
> plotMetaDataExpl(myPanel, "length", log=FALSE, join=FALSE, color=
+                        "blueviolet")
```

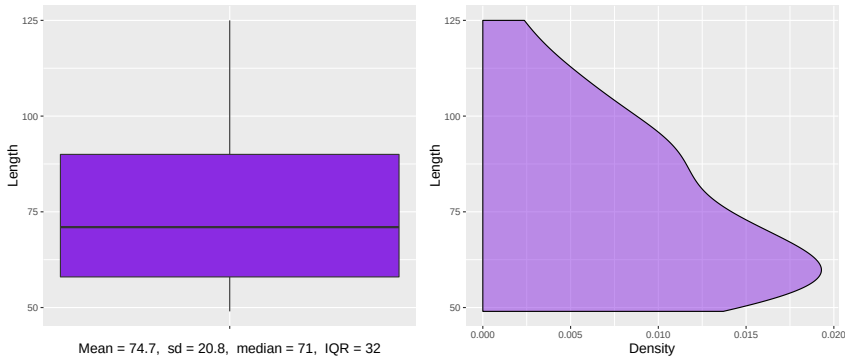


Figure 3: Amplicon length distribution' plot.

```
> # explore gene's relative frequencies
> plotMetaDataExpl(myPanel, "gene", abs=FALSE)
```

Figure 3 indicates that the mean amplicon length is 74.7 nucleotides with standard deviations of 20.8. But, as can be observed in the density plot, the mode is lower than this mean. In addition, half of the amplicons have their lengths lower than 71, and the rest between 71 and 125.
Figure 4 shows the gene relative frequencies, in percentages. As can be viewed, more than the 22% of the amplicons belong to 'gene8' and approximately 21% belong to 'gene5'. On the other hand, both 'gene1' and 'gene3' have less than 5% of total amplicons.

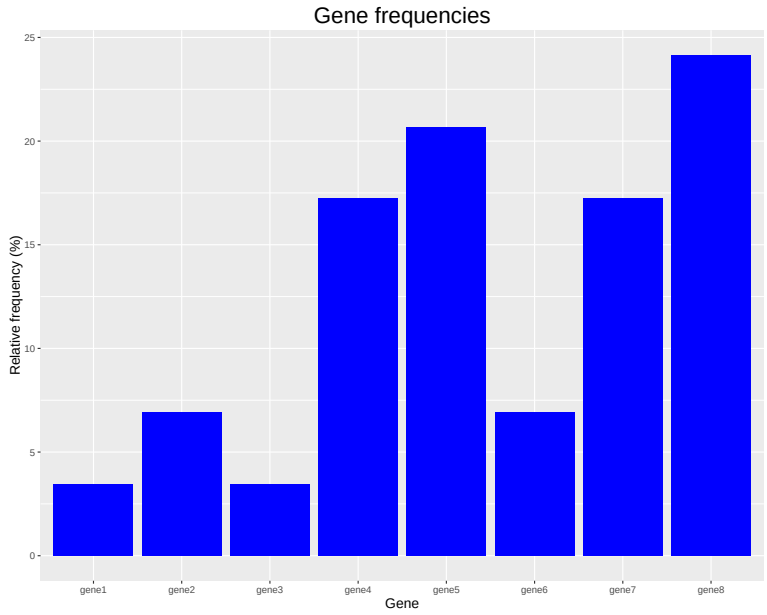


Figure 4: Gene's relative frequencies.

## 3.3 Deep exploration and Quality Control

### 3.3.1 Targeted selection performance

In the context of TS experiments, it is expected that most of the sequencing reads come from the target regions and not from the rest of the genome. If it does not, it may suggest an unexpected behavior of the used primers library that could impact on the read counts of overlapped regions skewing interpretation.
One way to check this is using the `readFrequencies` method which returns a *data frame* containing, for each chromosome, the amount and the percentage of reads fall in and out features. Therefore, if this graph shows a high percentage of reads that fall outside of interest features, a large part of the sequencing reads will be discarded when feature coverage is computed and consequently the power of detection will be lower. After `readFrequencies` calling, the `plotInOutFeatures` method can be used in order to plot the obtained *data frame*. It can be achieved running the next code:

```
> readFrequencies(myPanel)
> plotInOutFeatures(readFrequencies(myPanel))
```

Figure 5 shows that more than 15to genomic regions outside of the expected amplicons (red bars). This result is not expected, since it indicates that the used primers were not so specifics to capture only the targeted regions. Moreover, TS libraries are carefully designed to sequence specific genomic regions and the loss of a lot of sequence reads has a direct impact on the achieved coverage of the targeted regions. For these reasons the early detection of this technical effect can avoid false subsequent analysis conclusions.

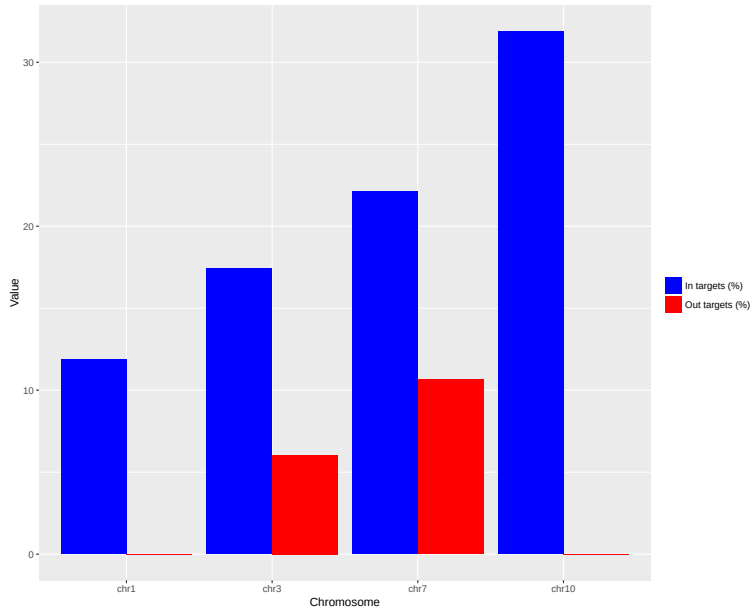


Figure 5: Percentages of reads falling in or out of the targeted regions.

### 3.3.2 Panel overview

Working on a TS experiment, it would be useful simultaneously evaluate the overall performance of the features. For instance, the user could be interested in exploring the proportion or amount of features within specific attribute intervals. In the example data, five coverage intervals could be defined according to the Table 1. Each user can define its attribute intervals table and the amount of them according to its needs.
In this example, 0 indicates *no reads* and *Inf* refers to features with coverage higher than 500. To incorporate

Table 1: Coverage intervals

| **Coverage Interval** | **Motivation** |
|---|---|
| $[0, 1)$ | *Not sequenced* |
| $[1, 50)$ | *Low sequencing coverage* |
| $[50, 200)$ | *Regular sequencing coverage* |
| $[200, 500)$ | *Very good sequencing coverage* |
| $[500, Inf)$ | *Excellent sequencing coverage* |

the coverage intervals into the analysis it is necessary to specify each interval extreme value like:

```
> # definition of the interval extreme values
> attributeThres<-c(0,1,50,200,500, Inf)
```

A panel results overview is critical in order to compare and integrate those at the feature, gene, and chromosome level. To help this task, *TarSeqQC*package implements the `plot` method, a graphical tool consisting in a polar histogram where each gene is represented as a bar. Each bar is colored depending on the percentage of features that have their attribute value within a particular prefixed interval. In addition, the bars (genes)

can be grouped in chromosomes in order to facilitate coverage results comparison at this level. The next code builds this plot:

```
> # plot panel overview
> plot(myPanel, attributeThres=attributeThres, chrLabels =TRUE)
```

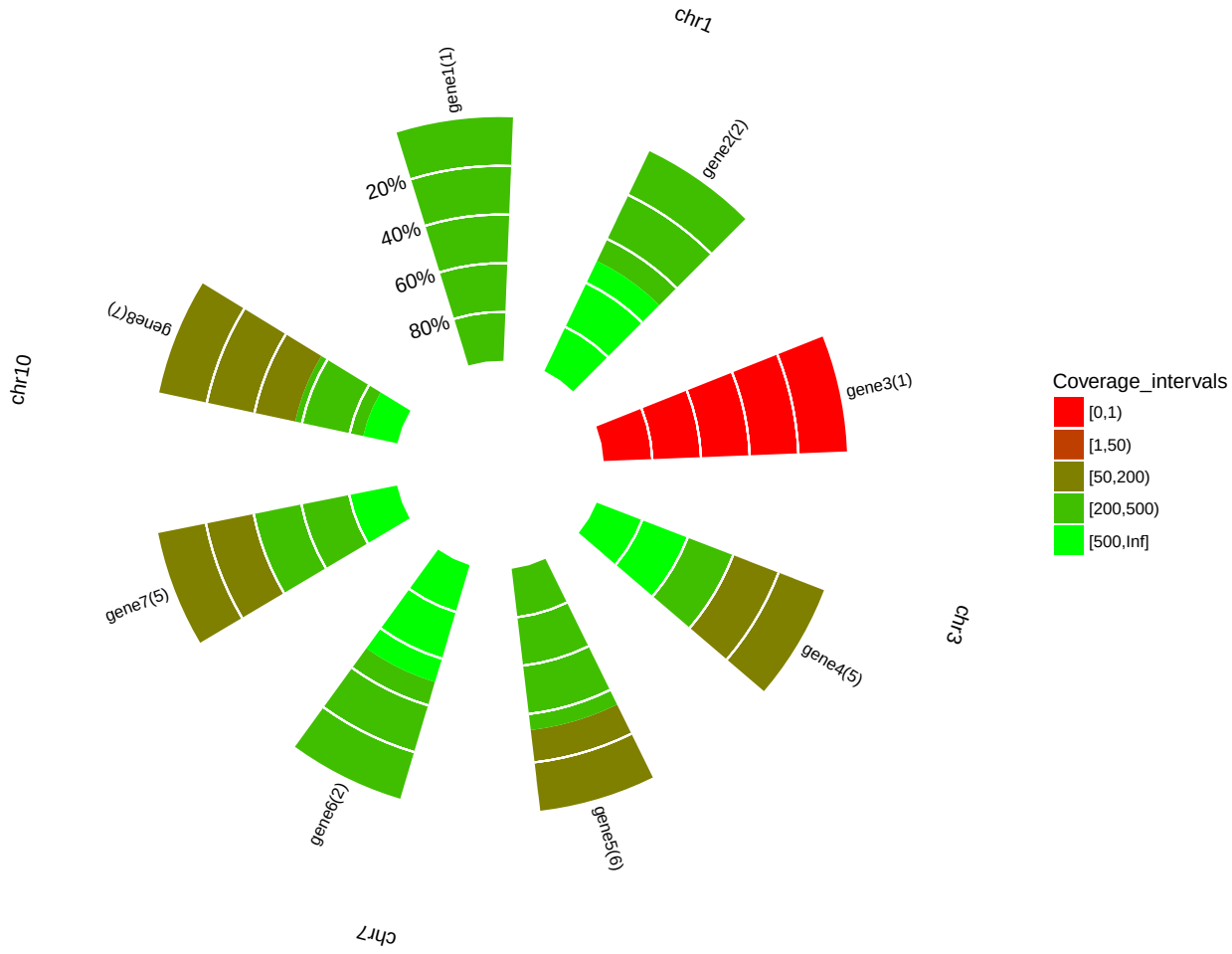


Figure 6: Panel overview plot.

In the example presented here, we can easily distinguish that the unique amplicon of the "gene3" was not sequenced. This is because in Figure 6, the bar corresponding to "gene3" is colored in red and this color is related to the [0,1) coverage interval. In the same plot, can also be appreciated that this gene has only one amplicon, as depicted in parenthesis in the bar label *"gene3(1)"*. Meanwhile "gene4" has 40% of its amplicons (2) with a coverage between 1 and 50, 20 % (1 amplicon) with coverage between 50 and 200 and the rest with a very good coverage value, greater than 200. It is notable that this plot provides a fast overview of the coverage feature performance.

It is important to note that a small and simple example is presented here. The previous plot could have a greater impact when the panel has more features and genes. Figure 7 shows the panel overview for a TS Experiment based on the *Ion AmpliSeq Cancer Panel Primer Pool* ®. This is a TS Panel offered by *Life Technologies* (Technologies (2014a)) that allows the inspection of 190 amplicons. In this case, can be easily observed that *MLH1* and *CDKN2A* genes were no sequenced. Can also be appreciated that several genes like *ALK, NRAS, BRAF, HNF1A*, have uniform coverage values along their amplicons. On the contrary, *RB1'* and *ATM* genes have some amplicons with low coverage and some other with a high coverage. Another alternative to explore attribute (coverage) performance is given by `plotFeatPerform` that shows a similar plot expanding it along the x-axis. This method also allows the exploration of attribute values at feature and gene levels, setting its parameter `complete` as `TRUE`. As result, the generated graph will contain two plots. The upper panel is a bar plot at the feature level and the lower, at the gene level. Both graphics incorporate the prefixed attribute intervals information and contain a red line to indicate the average value of the attribute at the corresponding level. In our example, you could run:

```
> # plot panel overview
> g<-plotFeatPerform(myPanel, attributeThres, complete=TRUE, log=FALSE,
+      featureLabs=TRUE, sepChr=TRUE, legend=TRUE)
> g
```

Figure 8 helps the coverage value evaluation for each amplicon and gene. It is noticeable that when coverage is summarized at gene level the highest value is lower than 500. However, at amplicon level, the highest value is greater than 800.
The previous plot is also very useful when TS panels were made-up by several primer pools combination. For example, the *Comprehensive Cancer Panel* ® is another *Life Technologies* panel that allows the exploration of 16000 amplicons from 409 genes related to several cancer types using 4 primer pools (Technologies (2014b)). In this case, the *Bed File* contains a 'pool column that stores the pool number for each feature. This information will be conserved in the *TargetExperiment* object that was built from this panel.
In the *Quality Control* context, it is important to evaluate in early analysis stages, if some pool effect exists and if all pool results are comparable. For example, Figure 9 illustrates the use of the `plotFeatPerform` in the described case. Now, you can see that the graphic corresponding to the amplicon level shows a separation between amplicons according to its pool value. Note that the same plot at a gene level is not showed because the `complete` parameter was set to `FALSE`. It is important to emphasize that, if correspond, the pool information will be included in all methods of the *TargetExperiment* class. Thus, for example, when you call the `summary` function for a *TargetExperiment* object that has pool information, the output will contain statistic results for the amplicon level and for each pool separately.

### 3.3.3 Controlling possible attribute bias

It is known that those DNA fragments having high GC content or high length tends to be 'more sequenced' those with lower GC/length values. Then, GC content and feature's length can be considered as possible bias sources, in particular, when the feature is an exon or a transcript. In order to check it, *TarSeqQC* incorporates `biasExploration` that allows the inspection of the selected attribute ('coverage' in the example) in terms of groups or intervals of bias sources. To do this, the method defines four source groups according to the distribution quartiles of the selected source (GC content, length or any included in the panel metadata) and then assigns one group to each feature. After that boxplot and, optionally, density plot for each group are plotted in order to evaluate possible bias. If the selected source is a categorical variable, like pool or gene, each category will be considered as a group. For instance, the next lines allows the exploration of GC content effect:

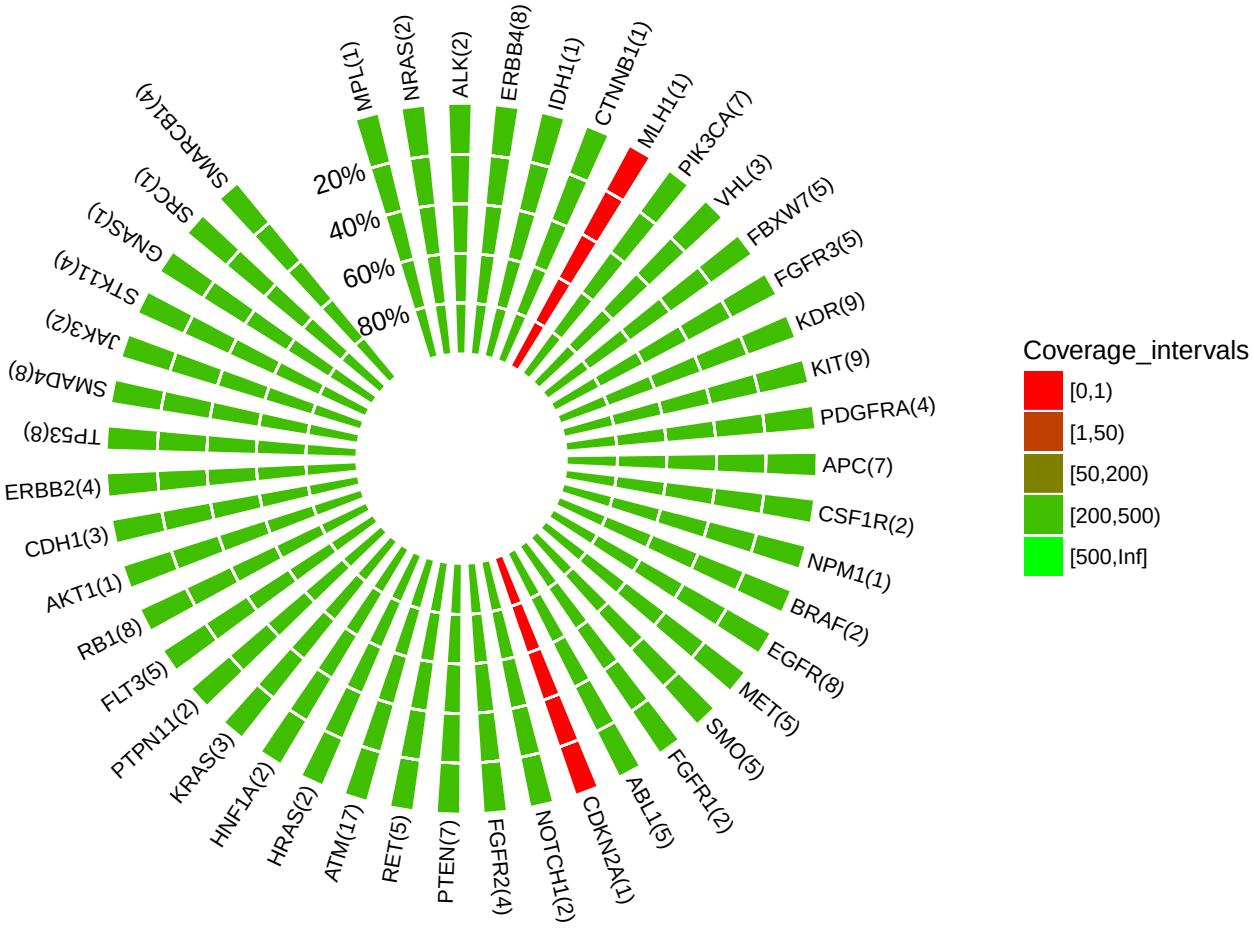




Figure 7: Cancer Panel Primer Pool overview plot.

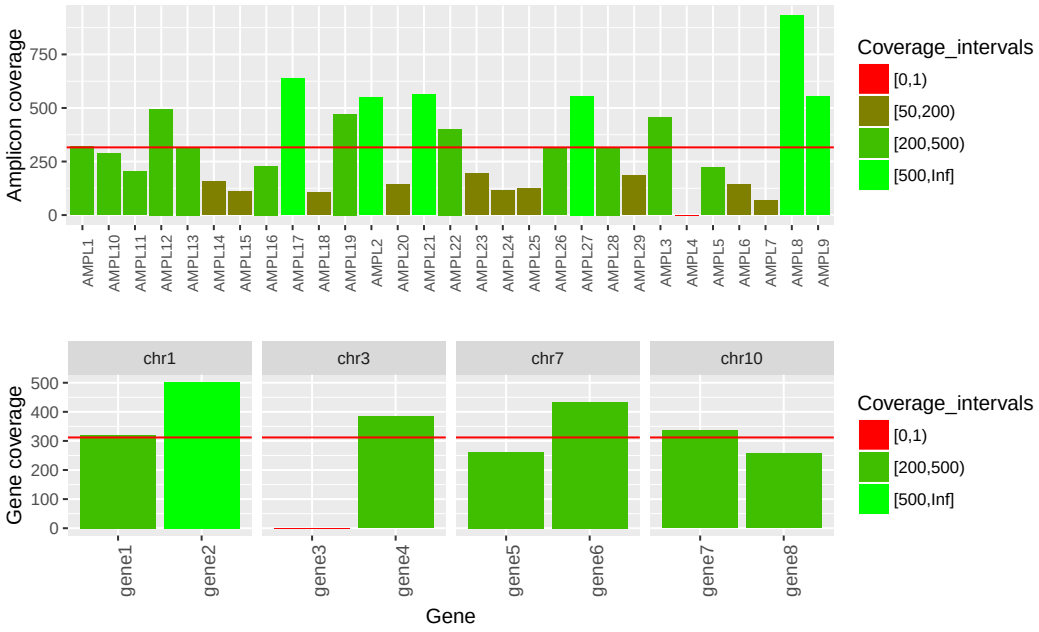


Figure 8: Amplicon coverage performance. The upper panel is a bar plot at feature level, and the lower, at gene level.

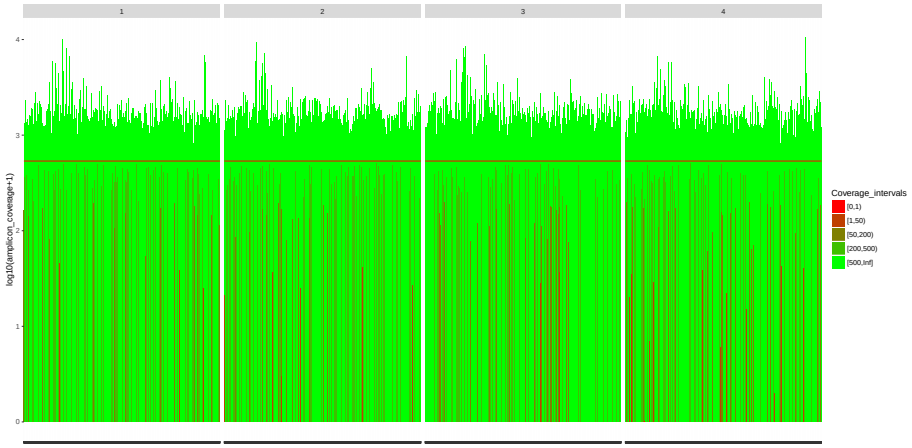


Figure 9: Performance exploration of an Ion AmpliSeq Comprehensive Cancer Panel experiment.

```
> x11(type="cairo")
> biasExploration(myPanel, source="gc", dens=TRUE)
```

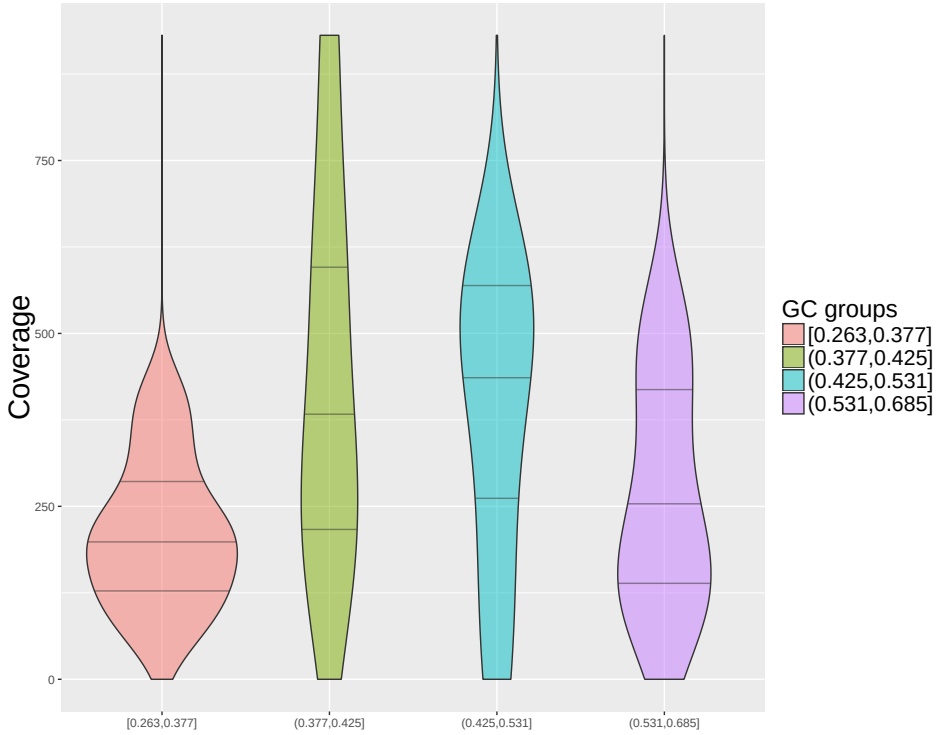


Figure 10: Exploration of amplicon coverage in each of amplicon GC content's quartiles.

### 3.3.4 Controlling low counts features

Low counts features should be detected in early analysis stages. The `summaryIntervals` method builds a frequency table of the fetaures that have its attribute value between predefined intervals. For example, if you are interested in exploring the "coverage" intervals defined before, you could do:

```
> # summaryIntervals
> summaryIntervals(myPanel, attributeThres)
```

```
  amplicon_coverage_intervals abs cum_abs  rel cum_rel
1       0  <=  coverage  <  1   1       1  3.4     3.4
2       1  <=  coverage  <  50   0       1  0.0     3.4
3      50  <=  coverage  <  200  10      11 34.5    37.9
4     200  <=  coverage  <  500  12      23 41.4    79.3
5          coverage  >=  500   6      29 20.7   100.0
```

The previous method is also useful when you are interested in quantifying how many features have its attribute value (*coverage*) lower or higher than a threshold. In this example, we are interested in knowing how many amplicons have shown at least a coverage of 50, because we consider that this is a minimum value that we will admit. This is a typical aspect that will be analyzed when you do an experiment *Quality Control*. Complementing this method, the `plotAttrPerform` method allows the graphical exploration of relative and cumulative feature frequencies in the predefined intervals. The corresponding function call is:

```
> plotAttrPerform(myPanel, attributeThres)
```

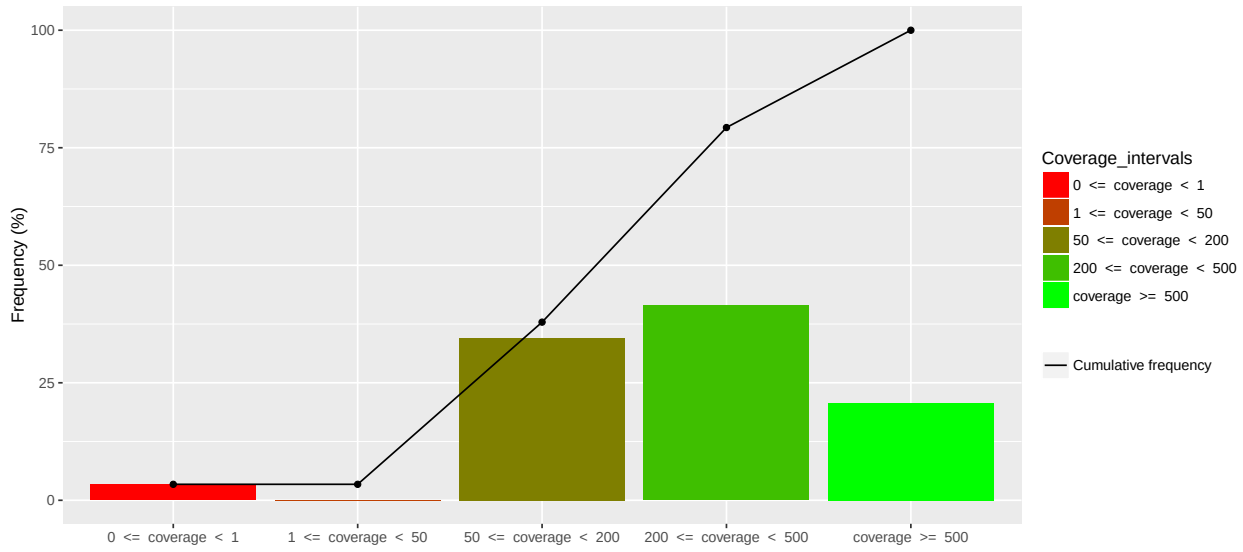


Figure 11: Relative and cumulative amplicon frequencies in the specified intervals.

In the cases in which several pools were used, both `summaryIntervals` and `plotAttrPerform` allows the exploration at pool level. Thus, differences among pool can be easily detected.
Another method that could help this analysis is `getLowCtsFeatures`. This method returns a *data frame* object containing all the features that have its attribute value lower than a threshold. The output *data frame* also contains panel and attribute information for each feature. For example, to known which are the genes that have a coverage value lower than 50, you can do:

```
> getLowCtsFeatures(myPanel, level="gene", threshold=50)
```

```
      names seqname start end medianCounts IQRCounts coverage sdCoverage
gene3 gene3    chr3     1  59            0         0        0          0
```

In addition, if you want to known which amplicons have a coverage value lower than 50, you should execute:

```
> getLowCtsFeatures(myPanel, level="feature", threshold=50)
```

```
      names seqname start end  gene    gc coverage sdCoverage medianCounts IQRCounts
AMPL4 AMPL4    chr3     1  59 gene3 0.492        0          0            0         0
```

Graphical methods were also implemented. The `plotGeneAttrPerFeat` allows the attribute value exploration for all the features of a selected gene. For instance, if you want to explore the "gene4", you should do:

```
> g<-plotGeneAttrPerFeat(myPanel, geneID="gene4")
> # adjust text size
> g<-g+theme(title=element_text(size=16), axis.title=element_text(size=16),
+      legend.text=element_text(size=14))
> g
```

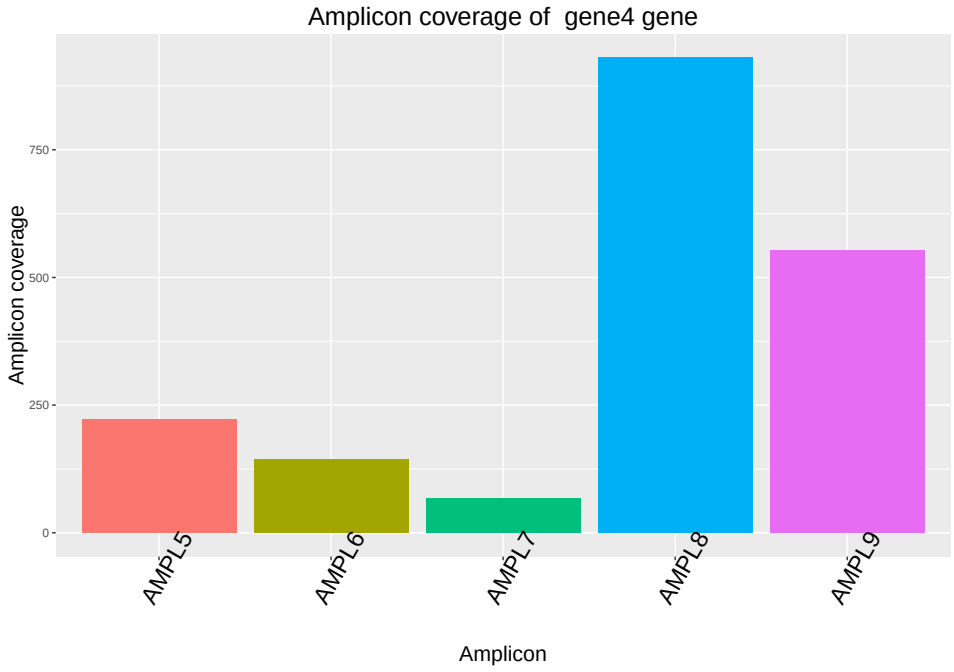


Figure 12: Performance attribute exploration of the *gene4*.

The attribute value for each feature contained in the "gene4" can be observed in Figure 12. If the selected gene has some features overlapped, the `overlap` and `level` parameters of the `plotGeneAttrPerFeat` method could be useful. When the `overlap` parameter is TRUE the method determines the overlap between features and defines new features called 'Regions'. For each of those regions, their attribute value is defined as the mean value of the attributes of the overlapped features. The `level` parameter, indicates the level of the output plot of the `PlotGeneAttrPerFeat`. The allowed values for `level` are: `"feature"`, `"region"` and `"both"`. The combination of these two parameters define the plot behavior:

- `overlap`=`FALSE`: amplicon overlap will be ignored.
    1. `level` must be `"feature"` (default values)
- `overlap` = `TRUE`: Amplicons will be grouped into regions according to their overlap
    1. `level` =`"feature"`, feature coverage bar plot grouped by overlapped regions as facets.
    2. `level` =`"region"`, overlapped region's coverage bar plot.
    3. `level` =`"both"`, plot with two paneles: the bar plot at feature level and at the region level.

For instance, in the last case, `overlap=TRUE` and `level="both"` the returned plot will consist of two panels. The top panel is the same that is generated when the `level` is `"feature"` including ggplot2 facets indicating which features are included in each region. The bottom panel is the returned when `level` is `"region"`.

```
> g<-plotGeneAttrPerFeat(myPanel, geneID="gene4", overlap = TRUE, level="both")
> g
```

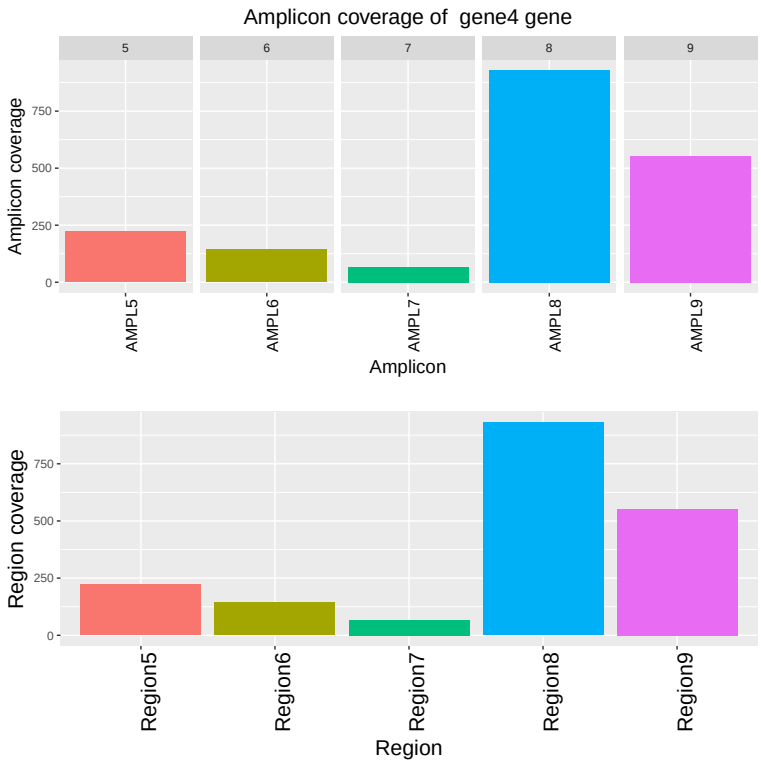


Figure 13: Performance attribute exploration of the *gene4* at amplicon and overlapped region levels.

It is worth to note that if a quality control at overlapped regions level is required, the user could define a new bed file in which the features will be the regions and run the TarSeqQC over this new bed file. To assist this task, the package provides the `getOverlappedRegions` method, internally called by the `plotGeneAttrPerFeat` method. `getOverlappedRegions` needs two parameters, a *TargetExperiment* object and a *logical* object, `collapse` indicating if the overlap information should be collapsed or not at the collapsed region level. Thus, if the user wants to define a new *Bed File* based on the overlapped regions, he could run the code below to obtain a *data.frame* summarizing the feature panel at a region level. Then, this object can be saved to a *Bed File* for future use of *TarSeqQC*.

```
> dfRegions<-getOverlappedRegions(myPanel,collapse=TRUE)
> head(dfRegions)

   chr start  end  gene region_id region_coverage
1 chr1   463  551 gene1   Region1             320
```

```
2 chr1  1553 1603 gene2   Region2             550
3 chr1  3766 3814 gene2   Region3             455
4 chr3     1   59 gene3   Region4               0
5 chr3  1062 1125 gene4   Region5             222
6 chr3  5720 5788 gene4   Region6             143
```

```
> #change the region_id field name to consistency of the new bed file
> names(dfRegions)[5]<-"name"
```

```
> #save the new bed file
> write.table(dfRegions[,1:5], file="myRegions.bed", sep="\t", col.names=T,
+  row.names=F,quote=F)
```

### 3.3.5 Read counts exploration

*Quality Control* in TS experiments implies the analysis of coverage/median counts achieved for each feature. But, sometimes, the exploration of the read profile that was obtained for a particular genomic region or a feature could be interesting. For this reason, the *TarSeqQC* R package provides methods to help the exploration at a nucleotide resolution. Those methods are based on the *data frame* returned by the `pileupCounts` method. This contains the read counts information for each nuecleotide of the features contained in the *Bed File*. Note that the columns in the obtained *data frame* could change, depending on the `pileupP` parameter definition. In our case we are working with its default constructor and only the `maxdepth` parameter was modified. For this reason, the resultant *data frame* will contain one column for each nucleotide and one column ("-") storing deletion counts.
`pileupCounts` is a function, not a *TargetExperiment* method, thus it could be called externally to the class. In order to call it, the specification of several parameters is needed:

- `bed`: is a *GRanges* object that, at least, should have values in the `seqnames`, `start` and `end` slots.
- `bamFile`: is a *character* indicating the full path to the *BAM File*.
- `fastaFile`: is a *character* indicating the full path to the *FASTA File*.
- `scanBamP`: is a *ScanBamParam* object, that specifies rules to scan the *BamFile*. If it was not specified, its default constructor values will be used and then, the `which` parameter will be specified using the `bed` parameter.
- `pileupP`: is a *PileupParam* object, that specifies rules to build the *pileup*, starting from the *BamFile*. If it was not specified, the `pileupP` parameter will be defined using the constructor default values.
- `BPPARAM`: is a *BiocParallelParam* instance defining the parallel back-end to be used during evaluation (see (Morgan et al., 2015a)).

In order to work with the example data, it is necessary do:

```
> # define function parameters
> bed<-getBedFile(myPanel)
> bamFile<-system.file("extdata", "mybam.bam", package="TarSeqQC", mustWork=TRUE)
> fastaFile<-system.file("extdata", "myfasta.fa", package="TarSeqQC",
+                        mustWork=TRUE)
> scanBamP<-getScanBamP(myPanel)
```

```
> pileupP<-getPileupP(myPanel)
> #call pileupCounts function
> myCounts<-pileupCounts(bed=bed, bamFile=bamFile, fastaFile=fastaFile,
+                        scanBamP=scanBamP, pileupP=pileupP, BPPARAM=BPPARAM)
```

```
> head(myCounts)
```

```
     pos seqnames seq   A   C   G   T N   = -  which_label counts
1345 463     chr1   T   1   4   0 167 0 167 0 chr1:463-551    172
1346 464     chr1   A 169   0   1   2 0 169 0 chr1:463-551    172
1347 465     chr1   G   1   0 174   1 0 174 0 chr1:463-551    176
1348 466     chr1   T   0   1   1 175 0 175 0 chr1:463-551    177
1349 467     chr1   G   0   0 181   0 0 181 0 chr1:463-551    181
1350 468     chr1   C   0 181   0   0 0 181 0 chr1:463-551    181
```

The obtained *data frame* contains the *pileup* information. It can be used to build a *read profile* plot where the x-axis represents the genomic position and the y-axis, the obtained read counts. The `plotRegion` allows exploration of the read profile for a specific genomic region. The `getRegion` method extracts the information for a genomic region. For example, the code below returns a *data frame* with location information of "gene7" amplicons:

```
> #complete information for gene7
> getRegion(myPanel, level="gene", ID="gene7", collapse=FALSE)
```

```
   names seqname start  end  gene
1 AMPL18   chr10   141  233 gene7
2 AMPL19   chr10  1007 1079 gene7
3 AMPL20   chr10  4866 4928 gene7
4 AMPL21   chr10  6632 6693 gene7
5 AMPL22   chr10  8475 8527 gene7
```

```
> #summarized information for gene7
> getRegion(myPanel, level="gene", ID="gene7", collapse=TRUE)
```

```
                                  names seqname start  end  gene
1 AMPL18, AMPL19, AMPL20, AMPL21, AMPL22   chr10   141 8527 gene7
```

Then, the previous information can be used to specify a genomic region and plot its read count profile using the `plotRegion` method. Since a large number of little changes at nucleotide level could be coused by the sequencing process, the method provides a way to filter out those noisy variants of the read profile. The `minAAF` specifies the minimum alternative allele frequency need to call a SNP and its default value is 0.05. The other parameter is called `minRD` and specifies the minimum read depth of the alternative allele (defult value = 10). For each position of the specified region, the method computes a threshold that the SNPs should be exceed to be plotted. To compute this threshold the `minAAF` is traduced at the read count scale by multplication of it with the total counts of the genomic position. Then, the threshold is defined as the maximum value between `minAAF` (in read counts scale) and `minRD`. To avoid these filters both should be set to 0. For instance, the call of the `plotRegion` avoiding these filters is:

```
> g<-plotRegion(myPanel, region=c(4500,6800), seqname="chr10", SNPs=TRUE,
+       minAAF = 0, minRD = 0, xlab="", title="gene7 amplicons",size=0.5)
> x11(type="cairo")
> g
```

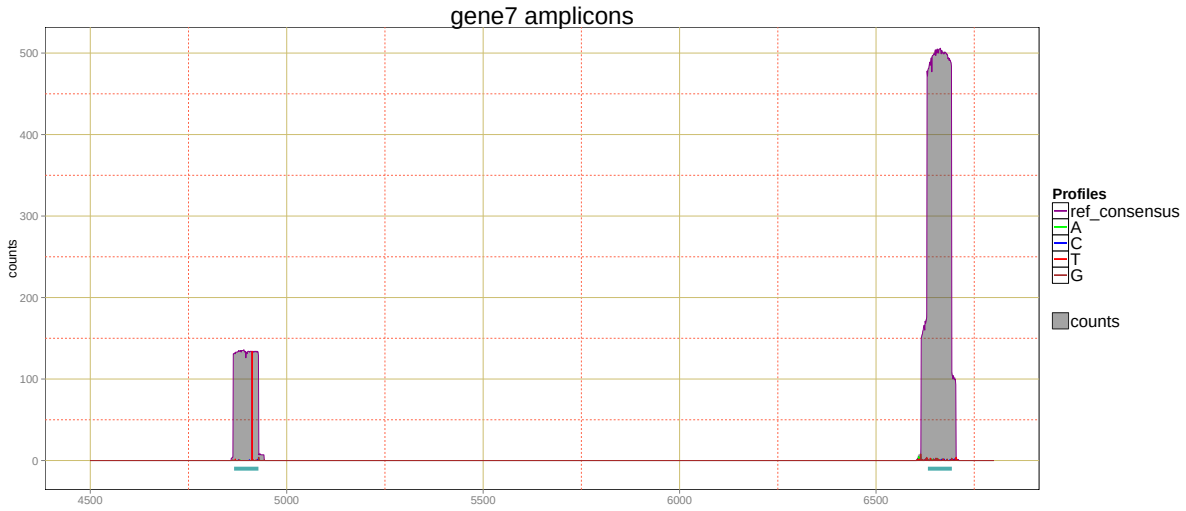


Figure 14: Read counts profile for the gene7 genomic region.

Another method, `plotFeature`, allows exploring the read profile of a particular feature. In this case,it is only necessary to specify the feature name that should be explored. This method internally calls the `plotRegion`, thus it also could use the `minAAF` and `minRD` parameters. For instance, the R code below illustrates the way to explore the "AMPL20" amplicon of the "gene7", considering only those genomics variations exceed `minAAF` =0.05 and `minRD`=10, which are the default values of these parameters.

```
> g<-plotFeature(myPanel, featureID="AMPL20")
> x11(type="cairo")
> g
```

As can be seen in both Figure 14 and Figure 15, the gray shadow corresponds to the total counts that were obtained at each genomic position insight the selected amplicon. The violet line indicates read counts matching with the reference sequence. In order to distinguish how many read counts could correspond to a genomic variation, it is crucial that the `pileupP` definition contains the `distinguish nucleotide` parameter as TRUE, which is its default value. In the previous plots, the green, blue, red and brown lines illustrate the read counts that do not match with the reference and inform about a possible nucleotide variation. In the Figure 15 can be appreciated that the selected amplicon shows a variation changing the reference nucleotide for a "T".
The proportion of read counts that match and no match against the reference can be displayed using the `plotNtdPercentage` method as:

```
> plotNtdPercentage(myPanel, featureID="AMPL20")
```

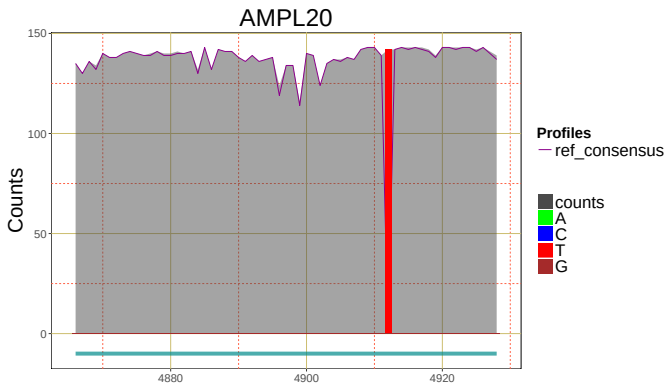


Figure 15: Read counts profile for the "20" gene7 amplicon.

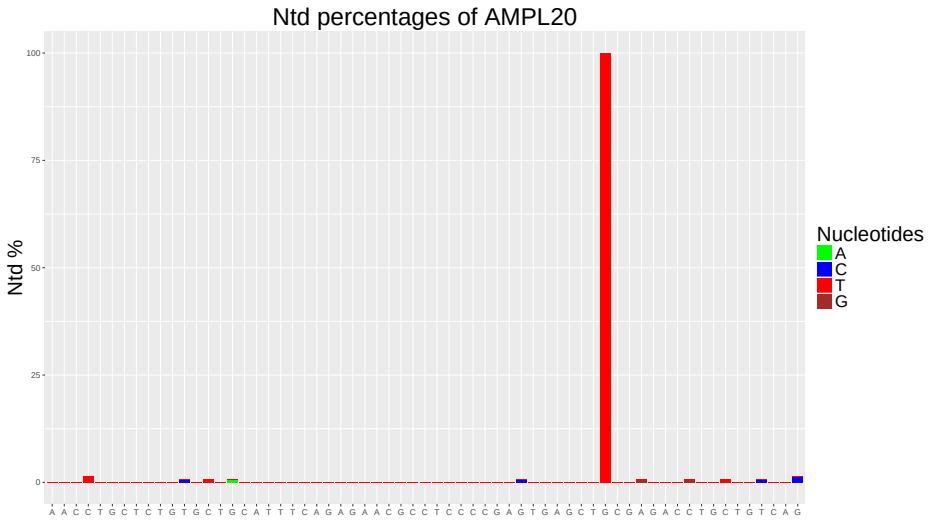


Figure 16: Nucleotide percentages for each genomic position on the "AMPL20" gene7 amplicon.

In Figure 16 can be observed that between 4910 and 4920 positions there is a nucleotide showing differences between reference and read sequences. This information can be also extracted using the previous read counts *data frame*, `myCounts` and the `featurePanel` slot of the *TargetExperiment*object. To do this only the feature name is necessary:

```
> getFeaturePanel(myPanel)["AMPL20"]
```

```
GRanges object with 1 range and 6 metadata columns:
         seqnames    ranges strand |        gene        gc  coverage sdCoverage
            <Rle> <IRanges>  <Rle> | <character> <numeric> <numeric>  <numeric>
  AMPL20     chr10 4866-4928      * |       gene7     0.587       142          1
         medianCounts IQRCounts
            <numeric> <numeric>
```

```
AMPL20          143         1
-------
seqinfo: 4 sequences from an unspecified genome; no seqlengths
```

Using this information, you could select a subset in the `myCounts` object containing only those nucleotide rows corresponding to the feature:

```
> featureCounts<-myCounts[myCounts[, "seqnames"] =="chr10" &
+                         myCounts[,"pos"] >= 4866 & myCounts[,"pos"] <= 4928,]
```

Then, the position having the lowest value in the "=" column can be found. The position with the minimum value of read counts matching against the reference is the same position that has the higher variation:

```
> featureCounts[which.min(featureCounts[,"="]),]
```

```
      pos seqnames seq A C G   T N = -     which_label counts
1423 4912    chr10   G 0 0 0 134 0 0 0 chr10:4866-4928    134
```

As can be observed, in the position 4912 the reference genome indicates that there should be a "G", and the read counts indicate that in this position there is a "T", showing a possible nucleotide variation. Remember that the presented analysis is only an exploratory tool. Thus, the possible nucleotide variations detected in this stage should be confirmed or rejected in more specific downstream analysis. In the same way, a variant identified in variant analysis can be easily explored and confirmed using our tool.

### 3.4 Quality Control Report

The *TarSeqQC* R package provides a method that generates an .xlsx report in which *Quality Control* relevant information is contained. This file has three sheets. In the first, a summary is presented, containing the results of `summary` and `summaryIntervals` methods. This sheet also includes a plot characterizing the experiment. Any graphic could be chosen, but if its name is not specified, the method calls the *TarSeqQC* `plot` method to build it. The second and third sheets store the panel information at a gene and a feature level respectively. Only the information corresponding to the selected attribute will be stored. Then, if only the report generation is desired, the `buildReport` method can be called after the object construction. In the present example, the image file that should be included in the report and the report creation is specified, using the code below:

```
> imageFile<-system.file("extdata", "plot.pdf", package="TarSeqQC",
+                         mustWork=TRUE)
> buildReport(myPanel, attributeThres, imageFile ,file="Results.xlsx")
```

## 4 The *TargetExperimentList* class

The *TargetExperimentList* class was built in order to allow the joint analysis of several *TargetExperiment* objects. Figure 17 shows the *TargetExperimentList* class structure.

The *TargetExperimentList* class has four slots:

- `bedFile`: a *GRanges* object that models the *Bed File*
- `panels`: a *GRanges* object that contains information related to several feature panels and related statistics.
- `attribute`: a *character* indicating which attribute *coverage* or *medianCounts* will be used to the analysis.
- `feature`: a *character* indicating the name of the analyzed features, e.g.: "amplicon", "exon", "transcript".

*TargetExperimentList* has implemented several methods that are an extension of previously described methods for *TargetExperiment* class. The next sections will illustrate the employment of *TargetExperimentList* methods using a *TargetExperimentList* object provided with the package. This dataset contains information of two synthetic amplicon sequencing experiments using 29 *amplicons* of 8 genes in 4 chromosomes and sequenced using two PCR pools.

## 4.1 Input Data

The *TargetExperimentList* allows the joint analysis of several TS experiments performed using the same *Bed File* that can be involved in several situations such as a disease characterization study, patient monitoring or system calibration process.
As a first step, one *TargetExperiment* object for each of the TS experiments should be built. Then, a *list* object, called `panelList`, should be defined where each of its elements will be a *TargetExperiment* object. For instance, having defined two *TargetExperiment* objects, `te1` and `te2`, the *list* must be defined as:

```
> panelList<-list(panel1=te1, panel2=te2)
```

After that, the *TargetExperimentList* constructor can be called passing three parameters: `panelList`, `feature` and `attribute`. The las two are the same previously used in panels definitions. It is worth highlighting

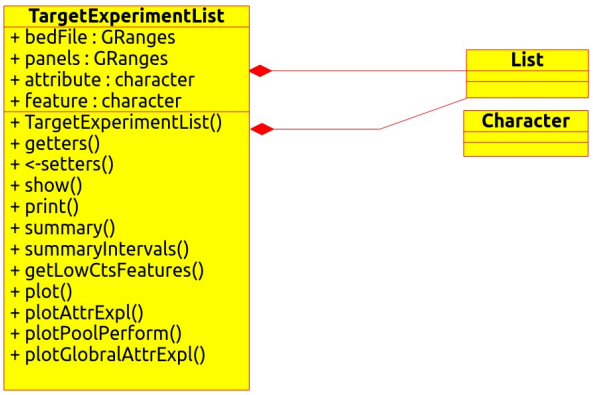


Figure 17: *TargetExperimentList* class diagram.

that in this case `attribute` cannot be specified after object definition. When the constructor is called, the resultant object only conserves information of the selected attribute. For instance, if the `attribute` is equal to 'coverage', the 'medianCounts' information will not be stored in the *TargetExperimentList* object. Thus, if the user wants to change the selected `attribute`, the *TargetExperimentList* constructor must be called again.

## 4.2 Creating a *TargetExperimentList* object

### 4.2.1 Constructor call

*TargetExperimentList* constructor can be now called using:

```
> TEList<-TargetExperimentList(TEList=ampliList, feature="amplicon",
+      attribute="coverage")
```

This object is provided with the *TarSeqQC* package and can be loaded doing:

```
> data(TEList, package = "TarSeqQC")
```

### 4.2.2 Early exploration

As *TargetExperiment*, the *TargetExperimentList* class has also implemented typical R methods such as `show`, `print`, and `summary`. They can be used as:

```
> # show/print
> TEList
```

```
TargetExperimentList
amplicon panels:
        GRanges object with 3 ranges and 5 metadata columns:
        seqnames    ranges strand |        gene      pool        gc coverage_panel_1
           <Rle> <IRanges>  <Rle> | <character> <numeric> <numeric>        <numeric>
  AMPL1     chr1   463-551      * |       gene1         2     0.674              320
  AMPL2     chr1 1553-1603      * |       gene2         2     0.451              550
  AMPL3     chr1 3766-3814      * |       gene2         2     0.531              455
        coverage_panel_2
               <numeric>
  AMPL1                0
  AMPL2              202
  AMPL3              152
  -------
  seqinfo: 4 sequences from an unspecified genome; no seqlengths

selected attribute:
         coverage
```

```
> # summary
> summary(TEList)

$panel_1
                   Min. 1st Qu. Median Mean 3rd Qu. Max.
amplicon coverage    0     143    288  316     472  931

$panel_2
                   Min. 1st Qu. Median Mean 3rd Qu. Max.
amplicon coverage    0       8     38   74     135  260

$`pool 1`
                   Min. 1st Qu. Median Mean 3rd Qu. Max.
amplicon coverage    3      67    164  246     316  931

$`pool 2`
                   Min. 1st Qu. Median Mean 3rd Qu. Max.
amplicon coverage    0      10    113  153     208  550
```

The `summary` method returns a list where each element is a statistic summary table computed in each feature panel. In particular, if the TS involves several PCR primer pools, then the method will display statistics summaries for each panel/sample and for each pool as can be seen in the example presented here. This basic exploration allows the identification of the panel 2 as a poor performance panel in wich the mean coverage was just 74 whereas, in panel 1, amplicons achieved a mean coverage of 316. It is also observed that amplicons in pool 1 achieved higher mean coverage in contrast of amplicons in pool 2 in both panels.
A simple way to observe the described situation is provided by the previously used `plotAttrExpl` method but now in the *TargetExperimentList* object. Now, the use of the method is:

```
> x11(type="cairo")
> plotAttrExpl(TEList, dens=TRUE, join=FALSE, log=FALSE, pool=TRUE)
```

The previous method provides several options allowing different configurations of the displayed plot. The parameter `dens` is a logical that indicates if density plot should be included and, if it is set to `TRUE`, then the `join` flag indicates if density and box-plots should be plot together or not. The `pool` parameter is also a logical and if it indicates PCR pool presence (`pool = TRUE`), then the plot are built for each pool separately using the *ggplot2* facets facility. The last case is the showed in Figure 18.
In order to allow the same exploration but at a pool level, *TargetExperimentList* class implements the `plotPoolPerformance` method. This has the same parameters specifications that the previous method excepting, obviously, the `pool` flag.

```
> x11(type="cairo");
> plotPoolPerformance(TEList, dens=TRUE, join=FALSE, log=FALSE)
```

## 4.3 Comparative analysis and Quality Control

As in the case of the *TargetExperiment* class, attribute intervals such the listed in Table 1 can be included in the panel comparative analysis. For instance, the two previous *TargetExperimentList* methods could accept

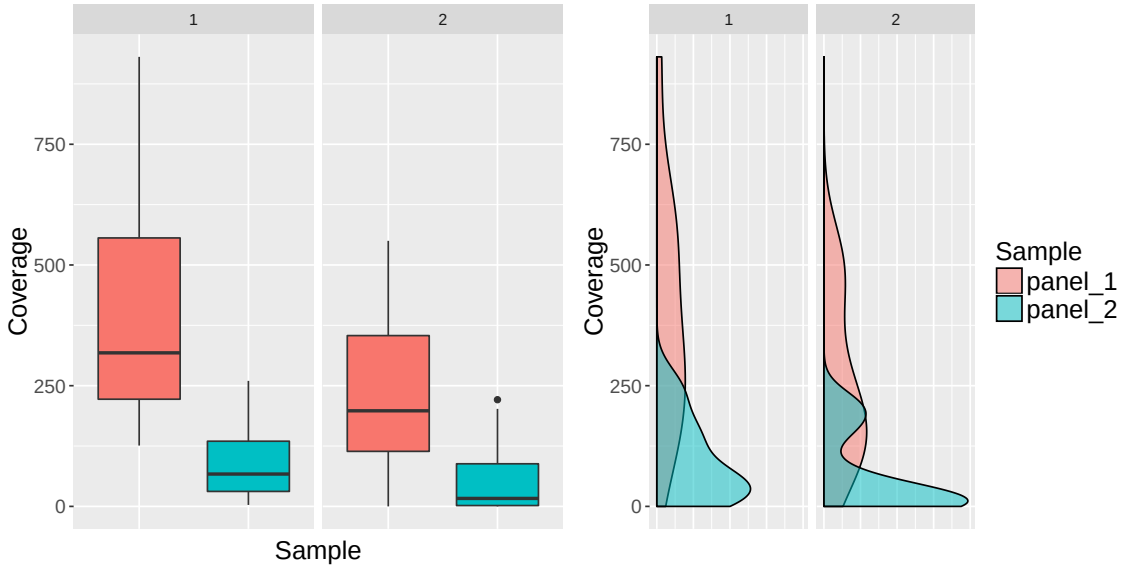


Figure 18: Attribute distribution and density plots for the two studied panels and separated by PCR pools.

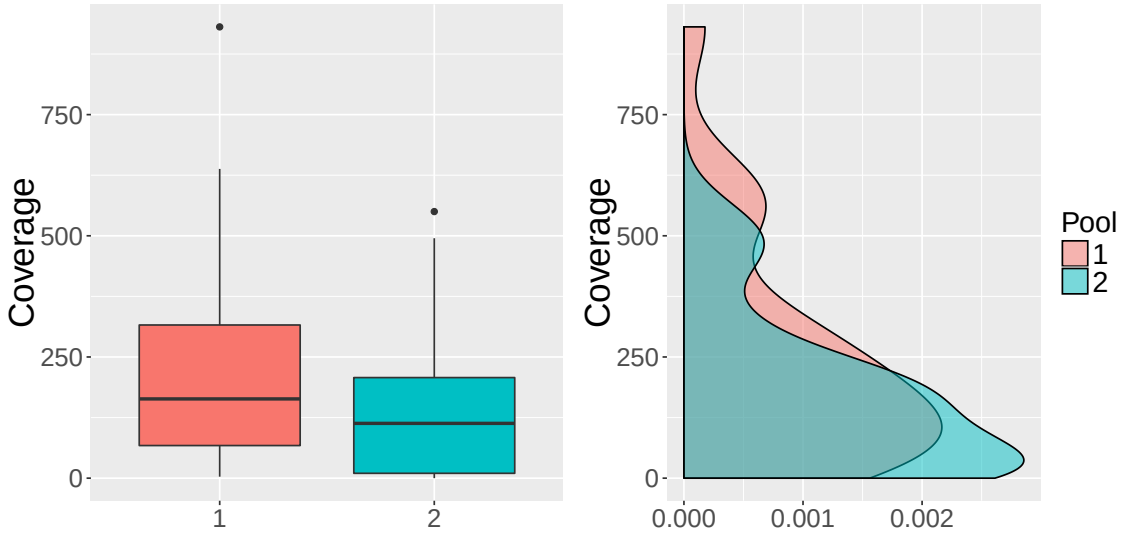


Figure 19: Attribute distribution and density plots for the two PCR pools.

a parameter `attributeThres` specifying the attribute intervals. If this parameter is specified, then the plots will be colored according to the interval in which fall the corresponding median value. The two previous figures were built without using this parameter and consequently, the plots shows one color for each sample. If in the corresponding methods calling `attributeThres` is added, then the resultant plots are showed in Figure 20 and Figure 21.

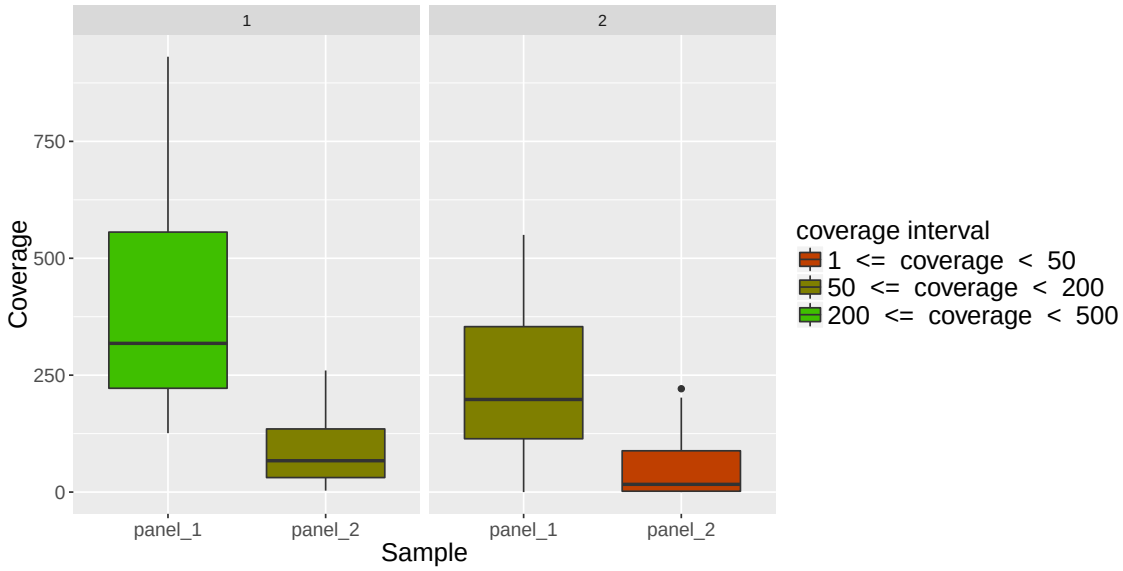


Figure 20: Attribute distribution and density plots for the two PCR pools.

The `plot` method was also implemented for *TargetExperimentList* class. In this case, the resultant *ggplot2* graph is a heatmap where rows represent features, columns represent samples and each cell is colored according to the attribute value for the feature in the corresponding sample. The method also allows the incorporation of pool information using *ggplot2* facets facilities. In the example case the follow lines generates this plot:

```
> plot(TEList, attributeThres = attributeThres, sampleLabs = TRUE,
+      featureLabs = TRUE)
```

The displayed heatmap confirms the behavior previously observed in `summary` results and in Figure 18 where panel 2 shows a boxplot with lower median compared with panel 1. In panel 2, more than 50 % of the amplicons have a coverage lower than 50 related to *low sequencing coverage* interval. Thus, if downstream analysis involves nucleotide variant identification, this panel should be discarded or re-sequenced in order to achieve higher coverage values that provide more credibility in variant identification.
When `featureLabs` is set to 'TRUE', the `plot` method allows the identification of features with low attribute value by simple inspection of x-axis ticks. If user wants to know not only who are the features with low attribute value but also what attribute value achieved each of these, thus `getLowCtsFeatures` can be used again as:

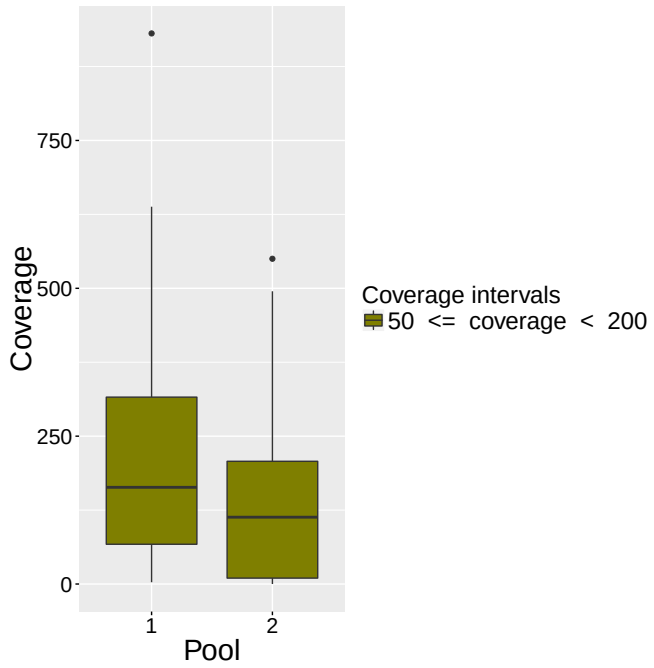


Figure 21: Attribute distribution and density plots for the two PCR pools.

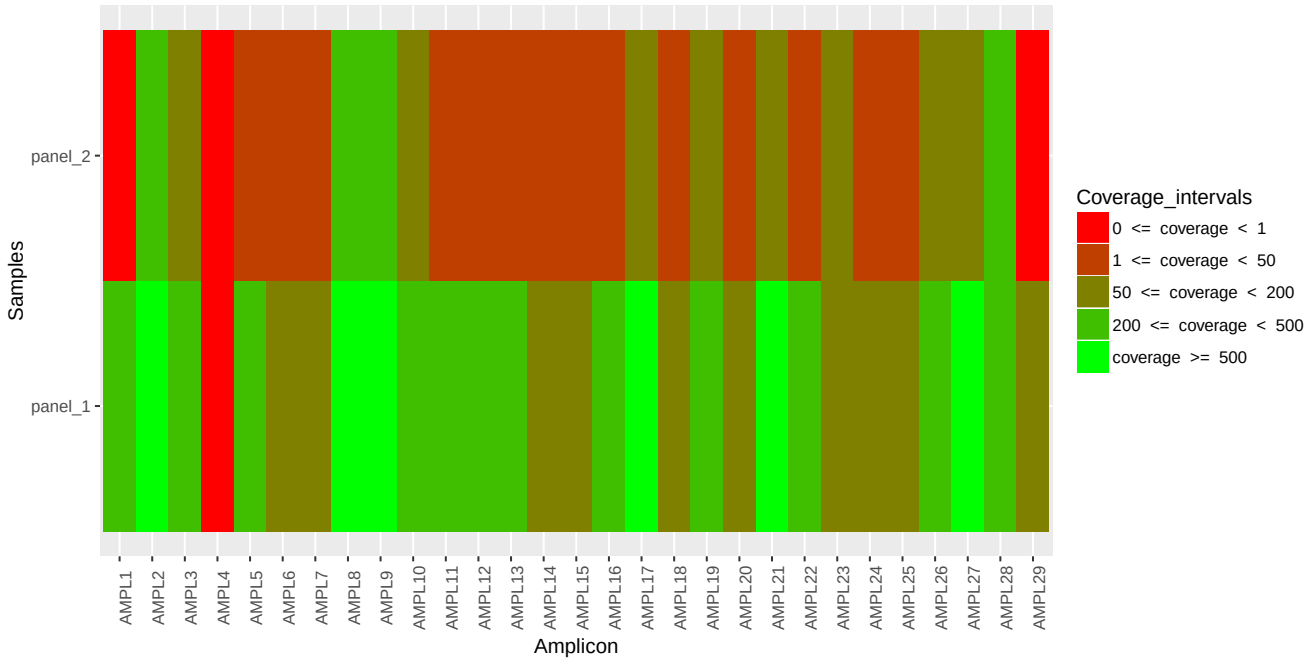


Figure 22: Panels overview: amplicons in rows and samples in columns. Colors according to prefixed coverage intervals

```
> getLowCtsFeatures(TEList, level="feature", threshold=50)
```

| | seqnames | start | end | width | strand | gene | pool | gc | coverage_panel_1 | coverage_panel_2 |
|---|---|---|---|---|---|---|---|---|---|---|
| AMPL4 | chr3 | 1 | 59 | 59 | * | gene3 | 2 | 0.492 | 0 | 0 |
| AMPL1 | chr1 | 463 | 551 | 89 | * | gene1 | 2 | 0.674 | 320 | 0 |
| AMPL5 | chr3 | 1062 | 1125 | 64 | * | gene4 | 1 | 0.359 | 222 | 3 |
| AMPL6 | chr3 | 5720 | 5788 | 69 | * | gene4 | 2 | 0.304 | 143 | 23 |
| AMPL7 | chr3 | 12149 | 12204 | 56 | * | gene4 | 2 | 0.268 | 67 | 2 |
| AMPL11 | chr7 | 10823 | 10893 | 71 | * | gene5 | 2 | 0.606 | 203 | 40 |
| AMPL12 | chr7 | 21989 | 22080 | 92 | * | gene5 | 2 | 0.598 | 495 | 25 |
| AMPL13 | chr7 | 30671 | 30744 | 74 | * | gene5 | 1 | 0.473 | 318 | 46 |
| AMPL14 | chr7 | 31443 | 31567 | 125 | * | gene5 | 1 | 0.464 | 158 | 38 |
| AMPL15 | chr7 | 38026 | 38098 | 73 | * | gene5 | 2 | 0.685 | 111 | 10 |
| AMPL16 | chr7 | 49567 | 49670 | 104 | * | gene6 | 2 | 0.356 | 229 | 8 |
| AMPL18 | chr10 | 141 | 233 | 93 | * | gene7 | 2 | 0.656 | 104 | 1 |
| AMPL20 | chr10 | 4866 | 4928 | 63 | * | gene7 | 1 | 0.587 | 142 | 14 |
| AMPL22 | chr10 | 8475 | 8527 | 53 | * | gene7 | 1 | 0.377 | 402 | 31 |
| AMPL24 | chr10 | 70550 | 70644 | 95 | * | gene8 | 2 | 0.263 | 115 | 10 |
| AMPL25 | chr10 | 97170 | 97220 | 51 | * | gene8 | 1 | 0.392 | 126 | 6 |
| AMPL29 | chr10 | 106083 | 106172 | 90 | * | gene8 | 2 | 0.278 | 184 | 0 |

The returned object is a *data.frame* containing those features or genes (depending on the value of the `level` parameter) that have attribute values lower than a threshold in at least one of the analyzed samples. For instance, the first amplicon appearing is 'AMPL4' having 0 coverage in both panels and the second amplicon is 'AMPL1' having coverage lower than 50 only in the second panel (0).
A more detailed view of the achieved attribute values per features is given for the `plotGlobalAttrExpl` *TargetExperimentList* method.

```
> x11(type="cairo")
> plotGlobalAttrExpl(TEList, attributeThres=attributeThres, dens=FALSE,
+      pool=TRUE, featureLabs=FALSE)
```

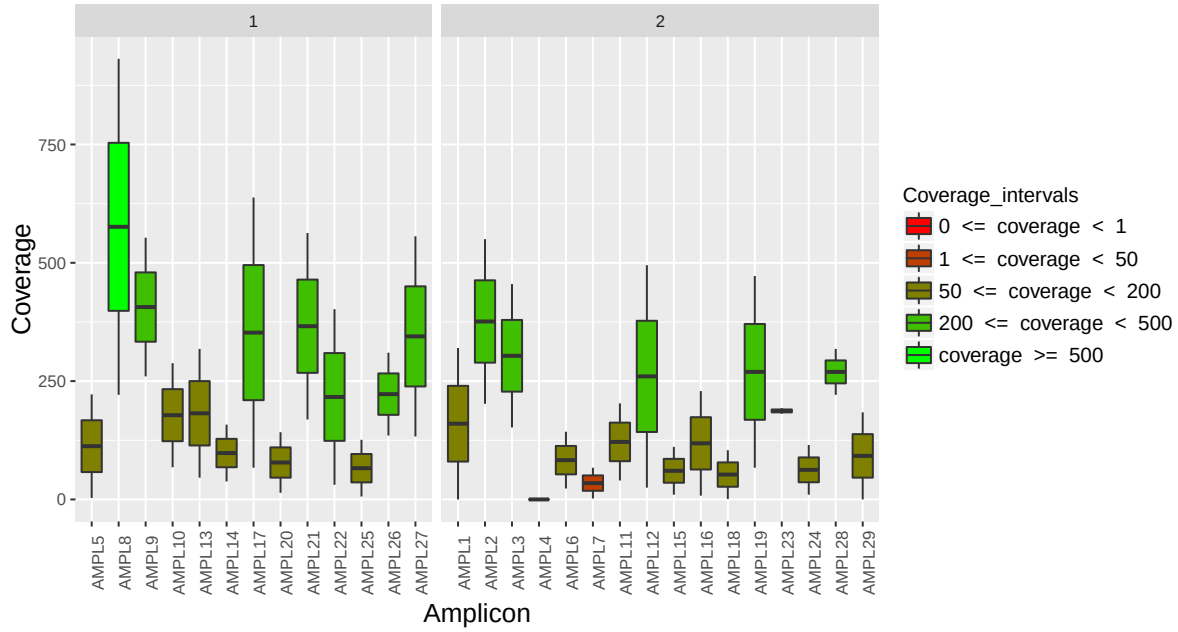


Figure 23: Amplicon performance. Each box-plot represents an amplicon, the values are the coverage achieved in the samples and its color is according to prefixed coverage intervals. Each facet represents a different PCR pool

Figure 23 shows the performance for each feature, in terms of achieved attribute, and allows the separation of them according to the PCR pool. Thus, the identification of poor performance pools is also allowed. In this case, both pool performances are similar and 'AMPL4' and 'AMPL7' show the worst performance.

# 5 Troubleshoot

`TargetExperiment` constructor in older TarSeqQC versions could result in an error message if the analyzed BAM file contains at least one unmapped read. For this, the BAM file section specify that the sorted alignment results should be used. Thus, unmapped reads need to be filtered before to theTarSeqQC use. To do this, the user could use the `scanBamP` parameter in the `TargetExperiment` constructor. The `scanBamP` is a *ScanBamParam* object that has several attributes. One of these is the `flag` which is a *scanBamFlag* that specifies the typical samtools flags used to scan a BAM file. Among the different options offered by the `scanBamFlag` constructor, the user can find the `isUnmappedQuery`. This is particularly useful to specify if the unmapped reads should be considered when the BAM file is scanned. Thus, if the user's BAM file could have any unmapped reads, we suggest the use of this parameter to ensure that those reads will not be considered when the `TargetExperiment` constructor is called. For instance, if it is suspected that the example BAM file contains some unmapped reads, the R code to successfully call the `TargetExperiment` constructor :

```
> bedFile<-system.file("extdata", "mybed.bed", package="TarSeqQC", mustWork=TRUE)
> fastaFile<-system.file("extdata", "myfasta.fa", package="TarSeqQC",
+      mustWork=TRUE)
> bamFile<-system.file("extdata", "mybam.bam", package="TarSeqQC", mustWork=TRUE)
> flag<-scanBamFlag(isUnmappedQuery = FALSE)
> scanBamP<-ScanBamParam(flag=flag)
> myPanel<-TargetExperiment(bedFile, bamFile, fastaFile, feature="amplicon",
+                           attribute="coverage", scanBamP=scanBamP,BPPARAM=BPPARAM)
>
```

Remember that all *TargetExperiment* methods that need read count information at a nucleotide level work over the *Bed File*, *BAM File* and the *FASTA File*. For this reason, in order to use some of them, please make sure that the corresponding *TargetExperiment* slots have the file names well defined. For example, if the *TarSeqQC* example data is loading as follows:

```
> data(ampliPanel, package="TarSeqQC")
> ampliPanel
```

```
TargetExperiment
amplicon panel:
        GRanges object with 3 ranges and 6 metadata columns:
        seqnames    ranges strand |        gene        gc  coverage sdCoverage
           <Rle> <IRanges>  <Rle> | <character> <numeric> <numeric>  <numeric>
  AMPL1     chr1   463-551      * |       gene1     0.674       320         19
  AMPL2     chr1 1553-1603      * |       gene2     0.451       550         90
  AMPL3     chr1 3766-3814      * |       gene2     0.531       455         12
        medianCounts IQRCounts
           <numeric> <numeric>
```

```
  AMPL1          326        24
  AMPL2          574        14
  AMPL3          463        27
  -------
  seqinfo: 4 sequences from an unspecified genome; no seqlengths

gene panel:
        GRanges object with 3 ranges and 4 metadata columns:
        seqnames    ranges strand | medianCounts IQRCounts  coverage sdCoverage
           <Rle> <IRanges>  <Rle> |    <numeric> <numeric> <numeric>  <numeric>
  gene1     chr1   463-551      * |          326         0       320          0
  gene2     chr1 1553-3814      * |          518        56       502         67
  gene3     chr3      1-59      * |            0         0         0          0
  -------
  seqinfo: 4 sequences from an unspecified genome; no seqlengths

selected attribute:
         coverage
```

and you want to explore a feature using the `plotFeature` method, the execution will cause an error because the method cannot find the files.

```
plotFeature(ampliPanel, featureID="AMPL1")
[1] "The index of your BAM file doesn't exist"
[1] "Building BAM file index"
open: No such file or directory
Error in FUN(X[[i]], ...) : failed to open SAM/BAM file
file: './mybam.bam'
```

To solve the previous error, the next code should be executed before:

```
> setBamFile(ampliPanel)<-system.file("extdata", "mybam.bam", package="TarSeqQC",
+                                     mustWork=TRUE)
> setFastaFile(ampliPanel)<-system.file("extdata", "myfasta.fa",
+                                     package="TarSeqQC", mustWork=TRUE)
```

and then:

```
> plotFeature(ampliPanel, featureID="AMPL1")
```

## Session Info

```
> sessionInfo()

R version 3.6.1 (2019-07-05)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 18.04.3 LTS

Matrix products: default
BLAS:   /home/biocbuild/bbs-3.10-bioc/R/lib/libRblas.so
LAPACK: /home/biocbuild/bbs-3.10-bioc/R/lib/libRlapack.so

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C               LC_TIME=en_US.UTF-8
 [4] LC_COLLATE=C               LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C                  LC_ADDRESS=C
[10] LC_TELEPHONE=C             LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

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

other attached packages:
 [1] BiocParallel_1.20.0  TarSeqQC_1.16.0      openxlsx_4.1.2       plyr_1.8.4
 [5] ggplot2_3.2.1        Rsamtools_2.2.0      Biostrings_2.54.0    XVector_0.26.0
 [9] GenomicRanges_1.38.0 GenomeInfoDb_1.22.0  IRanges_2.20.0       S4Vectors_0.24.0
[13] BiocGenerics_0.32.0

loaded via a namespace (and not attached):
 [1] Rcpp_1.0.2                  lattice_0.20-38             assertthat_0.2.1
 [4] digest_0.6.22               R6_2.4.0                    backports_1.1.5
 [7] acepack_1.4.1               pillar_1.4.2                zlibbioc_1.32.0
[10] rlang_0.4.1                 lazyeval_0.2.2              rstudioapi_0.10
[13] data.table_1.12.6           rpart_4.1-15                Matrix_1.2-17
[16] checkmate_1.9.4             splines_3.6.1               stringr_1.4.0
[19] foreign_0.8-72              htmlwidgets_1.5.1           RCurl_1.95-4.12
[22] munsell_0.5.0               DelayedArray_0.12.0         compiler_3.6.1
[25] xfun_0.10                   pkgconfig_2.0.3             base64enc_0.1-3
[28] htmltools_0.4.0             SummarizedExperiment_1.16.0 nnet_7.3-12
[31] tidyselect_0.2.5            tibble_2.1.3                gridExtra_2.3
[34] htmlTable_1.13.2            GenomeInfoDbData_1.2.2      matrixStats_0.55.0
[37] Hmisc_4.2-0                 crayon_1.3.4                dplyr_0.8.3
[40] withr_2.1.2                 GenomicAlignments_1.22.0    bitops_1.0-6
[43] grid_3.6.1                  gtable_0.3.0                magrittr_1.5
[46] scales_1.0.0                zip_2.0.4                   stringi_1.4.3
[49] reshape2_1.4.3              latticeExtra_0.6-28         cowplot_1.0.0
[52] Formula_1.2-3               RColorBrewer_1.1-2          tools_3.6.1
[55] Biobase_2.46.0              glue_1.3.1                  purrr_0.3.3
[58] survival_2.44-1.1           colorspace_1.4-1            cluster_2.1.0
[61] knitr_1.25
```

# References


Lawrence, M., Huber, W., Pagès, H., Aboyoun, P., Carlson, M., Gentleman, R., Morgan, M., and Carey, V. (2013). Software for computing and annotating genomic ranges. *PLoS Computational Biology*, 9.

Lee, H. C., Lai, K., Lorenc, M. T., Imelfort, M., Duran, C., and Edwards, D. (2012). Bioinformatics tools and databases for analysis of next-generation sequence data. *Briefings in functional genomics*, 11(1):12–24.

Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., Durbin, R., et al. (2009). The sequence alignment/map format and samtools. *Bioinformatics*, 25(16):2078–2079.

Metzker, M. L. (2010). Sequencing technologiesâĂŤthe next generation. *Nature Reviews Genetics*, 11(1):31–46.

Morgan, M., Obenchain, V., Lang, M., and Thompson, R. (2015a). *BiocParallel: Bioconductor facilities for parallel evaluation*. R package version 1.3.51.

Morgan, M., Pagès, H., Obenchain, V., and Hayden, N. (2015b). *Rsamtools: Binary alignment (BAM), FASTA, variant call (BCF), and tabix file import*. R package version 1.18.3.

Technologies, L. (2014a). Ion ampliseq cancer panel primer pool.

Technologies, L. (2014b). Ion ampliseq comprehensive cancer panel.