

TarSeqQC: Targeted Sequencing Experiment Quality Control

Gabriela A Merino¹, Cristóbal Fresno¹, and Elmer A Fernández¹

¹CONICET-Universidad Católica de Córdoba, Argentina

May 3, 2016

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 over 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.

Contents

1	Introduction	3
2	The <i>TargetExperiment</i> class	3
3	Input Data	5
3.1	Bed File	5
3.2	BAM File	6
3.3	FASTA File	6
3.4	Additional input data	7
4	Creating a <i>TargetExperiment</i> object	7
4.1	Constructor call	7
4.2	Early exploration	9
5	Deep exploration and Quality Control	11
5.1	Targeted selection performance	11
5.2	Panel overview	12
5.3	Controlling possible attribute bias	18
5.4	Controlling low counts features	18
5.5	Read counts exploration	20
6	Quality Control Report	25
7	Troubleshoot	26

1 Introduction

Next Generation Sequencing (NGS) technologies produce huge volume of sequence data at relative 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 Polimorphisms (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 a 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, 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 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 became 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 The *TargetExperiment* class

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

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.

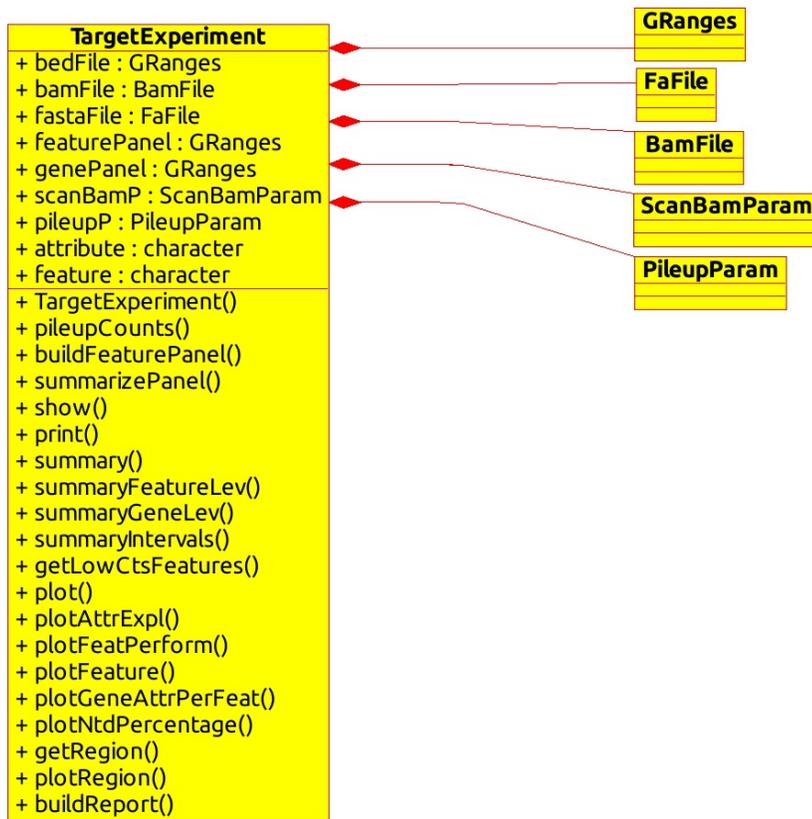


Figure 1: *TargetExperiment* class diagram.

- **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 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 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 as 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” 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 setted as 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”.

3.2 BAM File

The *BAM File* stores the 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 contains 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.

3.3 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 setted as a *TargetExperiment* slot.

3.4 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 setted as `""`. But, prior to perform 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 an 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 setted as `""`. But, as in the **attribute** parameter, it must be set prior to perform 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.

4 Creating a *TargetExperiment* object

4.1 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 contain 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 has 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 setted 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 in 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 file names is only necessary in order to use *TargetExperiment* methods that query this files.

4.2 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
   <Rle>      <IRanges> <Rle> | <character> <numeric> <numeric>
AMPL1    chr1 [ 463,  551]   * |    gene1    0.674    320
AMPL2    chr1 [1553, 1603]   * |    gene2    0.451    550
AMPL3    chr1 [3766, 3814]   * |    gene2    0.531    455
  sdCoverage medianCounts IQRCounts
   <numeric>   <numeric> <numeric>
AMPL1         19         326         24
AMPL2         90         574         14
AMPL3         12         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
   <Rle>      <IRanges> <Rle> |   <numeric> <numeric> <numeric>
gene1    chr1 [ 463,  551]   * |         326         0         320
gene2    chr1 [1553, 3814]   * |         518         56         502
gene3    chr3 [  1,   59]   * |          0          0          0
  sdCoverage
   <numeric>
gene1         0
gene2         67
gene3         0
-----
seqinfo: 4 sequences from an unspecified genome; no seqlengths
```

selected attribute:
coverage

```
> # summary
> summary(myPanel)
```

```
[1] "dfs"
      Min. 1st Qu. Median Mean 3rd Qu. Max.
gene      0     261   328  312   396  502
amplicon  0     143   288  316   472  931
      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 256. 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
```

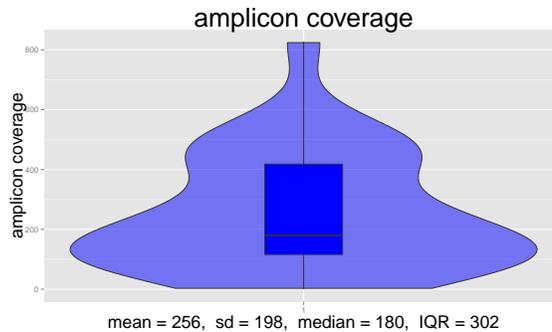


Figure 2: Attribute distribution and density plots.

In the Figure 2, the `join` parameter was set as 'TRUE'. Thus, density and box plots are plotted together. If it is set as '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.

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

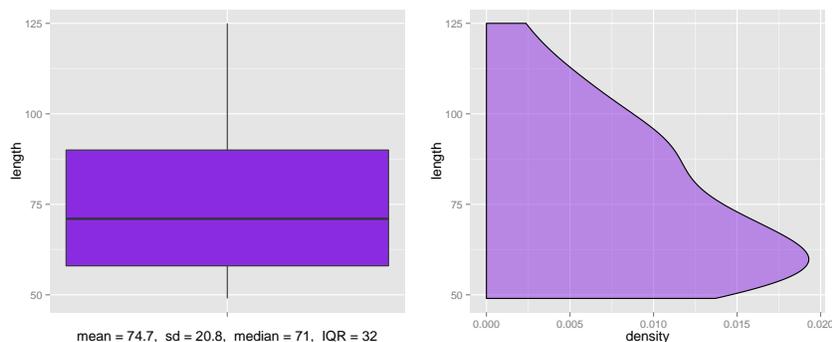


Figure 3: Amplicon length distribution' plot.

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

Figure 3 indicates that the mean amplicon length is 74.7 nucleotides with a 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.

The 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.

5 Deep exploration and Quality Control

5.1 Targeted selection performance

In the context of TS experiments, it is expected that most of sequencing reads come from the target regions and not the rest of the genome. If it does not, may be a resequencing using another primers will be

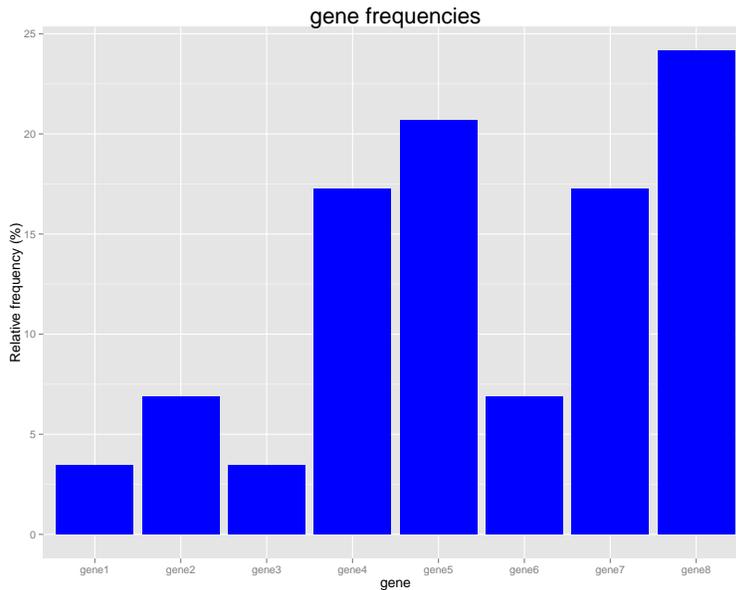


Figure 4: Gene’s relative frequencies.

needed. One way to check this is using the `readFrequencies` which returns a *data frame* containing, for each chromosome, the amount and the percentage of reads fall in and out features. Then, the `plotInOutFeatures` method can be used in order to plot this *data frame*. It can be achieved running the next code:

```
> readFrequencies(ampliPanel)
> plotInOutFeatures(readFrequencies(ampliPanel))
```

In the Figure 5 can be observed that more than half sequence reads belong to genomic regions outside of the amplicons (red bars). This is a not good results, because it indicates that the used primers were not so specifics to capture only the targeted regions.

5.2 Panel overview

When a person is working with a TS experiment, it is interesting to simultaneously evaluate the performance of all the features. In addition, if the user have prefixed attribute intervals, it could be important to compare features according to them. For example, five coverage intervals can be defined according to the Table 1.

Tthese coverage intervals could be incorporated into the analysis. To do this, the *TarSeqQC* R package needs an interval extreme definitions:

```
> # 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. To help this task, *TarSeqQC* package implements the `plot` method. This is a graphical tool consisting in a polar histogram, in which

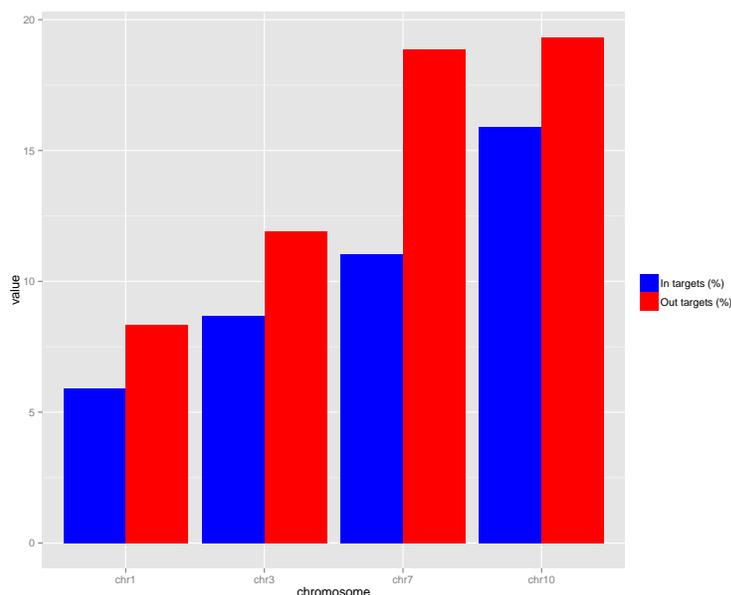


Figure 5: Percentages of reads falling in or out of the targeted regions.

Table 1: Coverage intervals

Coverage Interval	Motivation
$[0, 1)$	<i>Not sequenced</i>
$[1, 50)$	<i>Low sequencing coverage</i>
$[50, 200)$	<i>Regular sequencing coverage</i>
$[200, 500)$	<i>Very good sequencing coverage</i>
$[500, Inf)$	<i>Excellent sequencing coverage</i>

each gene is represented as a bar. Each bar is colored depending the percentage of features that have their attribute value in a particular prefixed interval. In addition, the bars (genes) can be grouped in chromosomes in order to facilitate the comparisson at this level. The next code help to build this plot:

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

In the example presented here, we can easily distinguish that the unique amplicon of the “gene3” was not sequenced. This is because in the 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)*”. Also it is possible to note that 40% of “gene4” amplicons has a coverage between 1 and 50. Note that this gene have five amplicons, then the 40 % corresponds to 2 amplicon. Another 20 % (1 amplicon) has a coverage value between 50 and 200, other one “gene4” amplicon have a very good coverage value, it means, between 200 and 500 and the other amplicon have an excellent coverage higher than 500.

It is important to note that a small and simple example is presented here. The prevoious plot could have

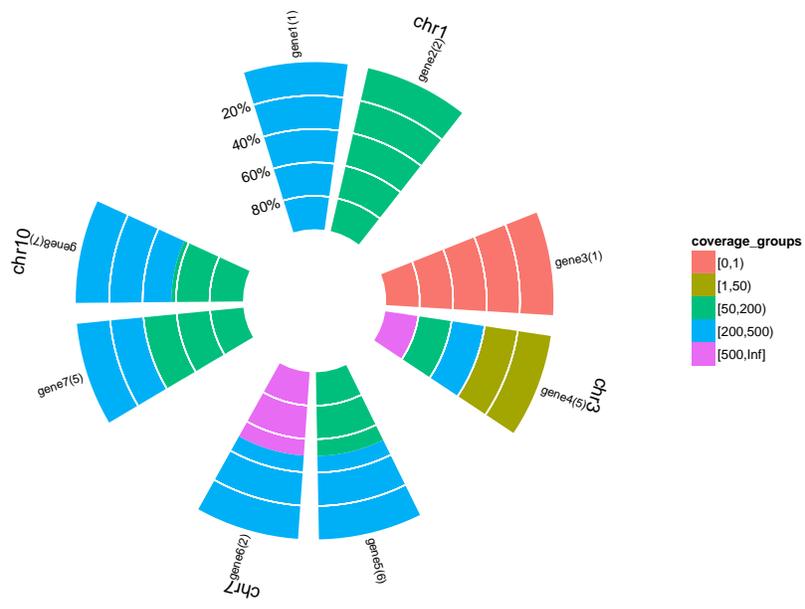


Figure 6: Panel overview plot.

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”, “VHL”, “AKT1”, “ARBB2”, have uniforme coverage values along their amplicons. On the contrary, “KDR” and “PTEN” genes have some amplicons that were not sequenced and some other with a high coverage. Complementing the previous plot, `plotFeatPerform` illustrates a similar graphic where

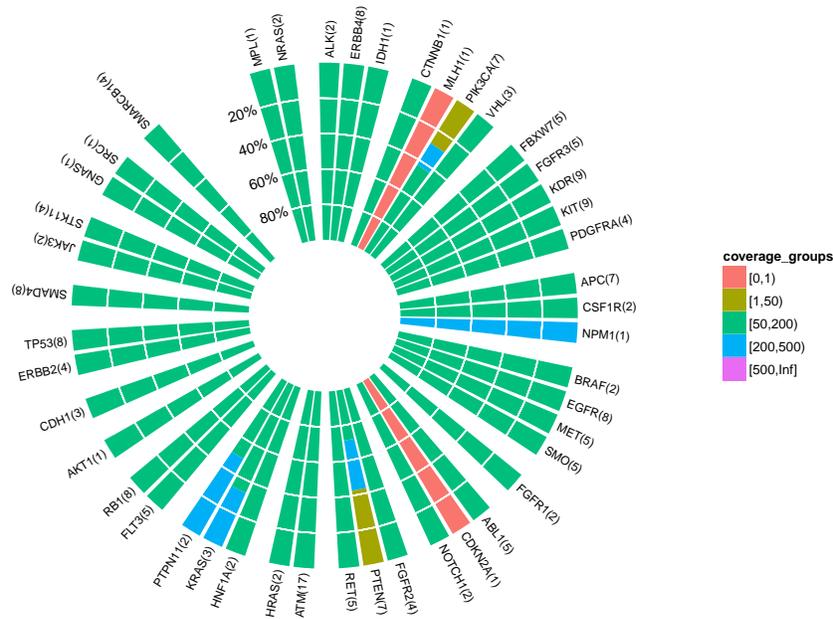


Figure 7: Cancer Panel Primer Pool overview plot.

the bars are distributed along the x-axis. In order to expand the polar histogram showed in Figure 6, the parameter `complete` is included. If you set it as `TRUE`, the resultant plot will contain two graphics. The upper panel is a bar plot at feature level, and the lower, at a 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
```

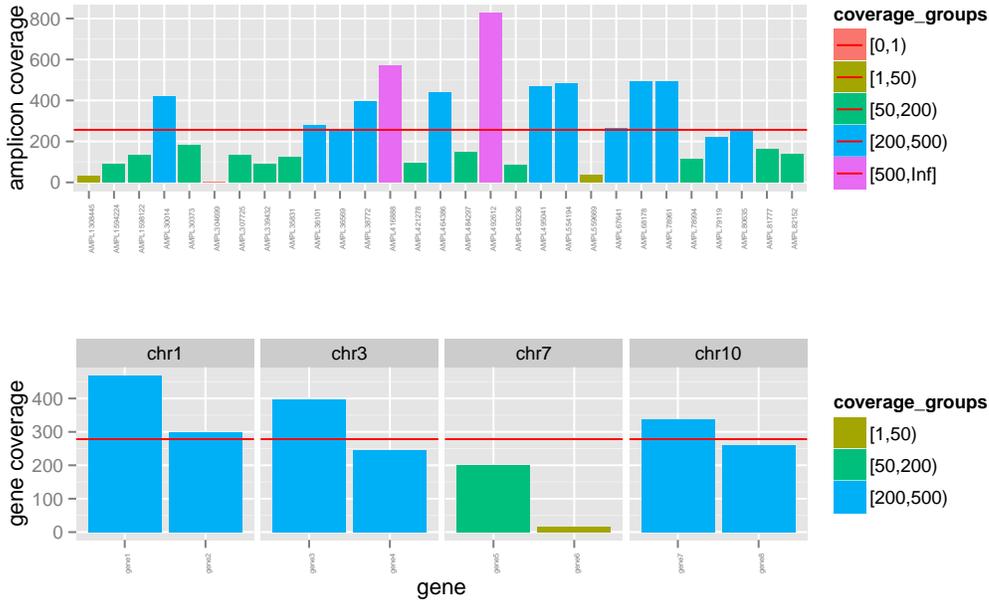


Figure 8: Amplicon coverage performance. The upper panel is a bar plot at feature level, and the lower, at gene level.

In Figure 8 we could evaluate the coverage value for each amplicon and gene. We can observe 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 we are working with panels 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 so important, evaluate in early analysis stages, if some pool effect exists and if all pool results are comparable. Naturally, the *TarSeqQC* R package uses this information to assist the user. For example, the 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.

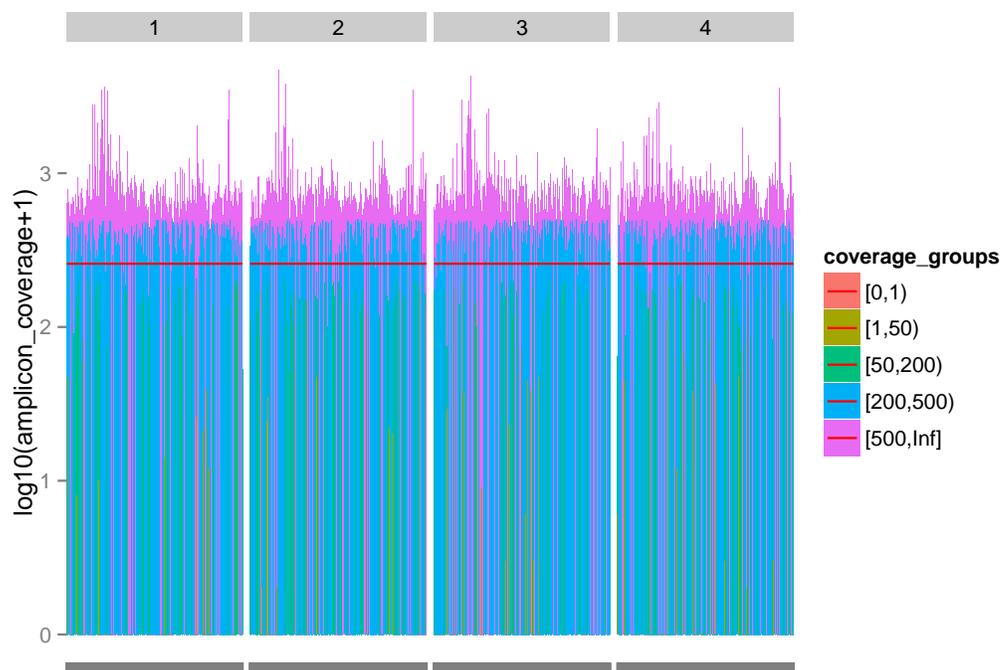


Figure 9: Performance exploration of an Ion AmpliSeq Comprehensive Cancer Panel experiment.

5.3 Controlling possible attribute bias

It is known that those genes having high GC content or high length tends to be 'more sequenced' than those with lower GC/length values. Then, GC content and feature 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 source's distribution quartiles and then assigns one group to each feature. After that boxplot and, optionally, density plot for each group are plotted in order to appreciate possible source bias. It can be performed for GC content, feature length, or any other metadata column. If the 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:

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

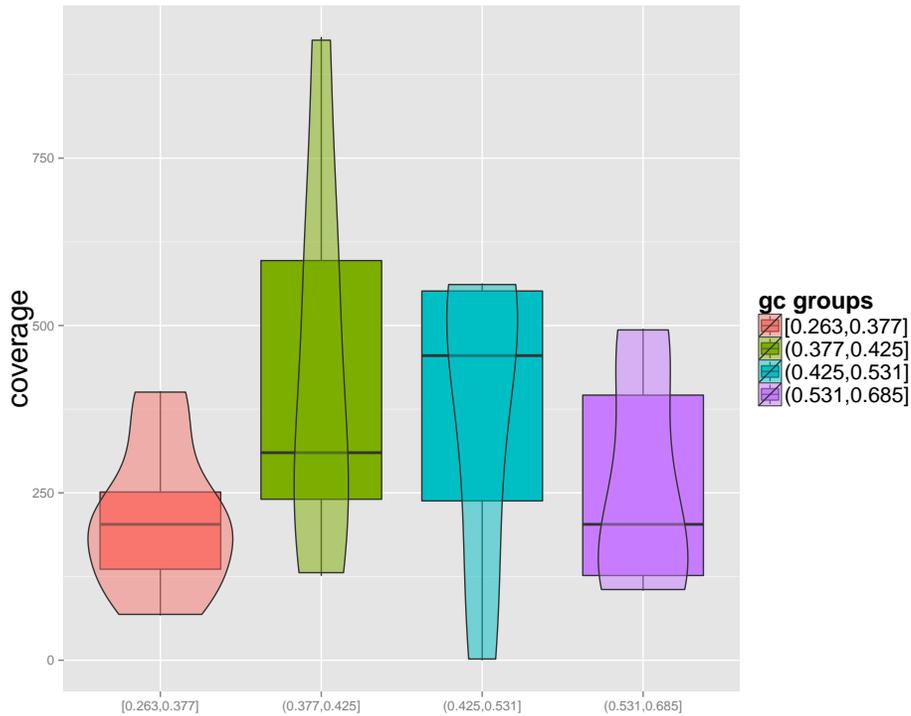


Figure 10: Exploration of amplicon coverage in each of amplicon GC content's quartiles.

5.4 Controlling low counts features

Low counts features should be detected in early analysis stages. The `summaryIntervals` method builds a frequency table of the features that have its attribute value between predefined intervals. For example, if you

are interested in explore 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 methods is also useful when you are interesting 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)
```

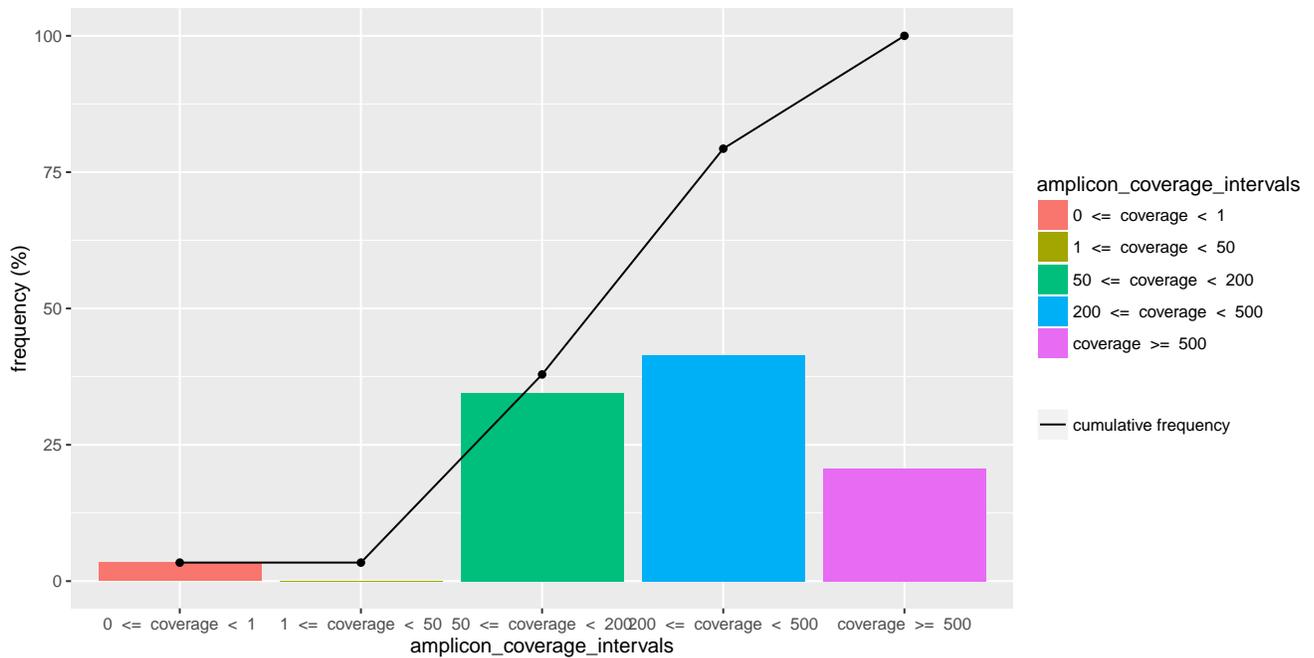


Figure 11: Relative and cumulative amplicon frequencies in the specified intervals.

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 know 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
1 gene3   chr3     1  59             0         0           0           0
```

In addition, if you want to know 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
1 AMPL4   chr3     1  59 gene3 0.492         0           0           0
  IQRCounts
1           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
```

The attribute value for each feature contained in the “gene4” can be observed in Figure 12.

5.5 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 nucleotide 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:

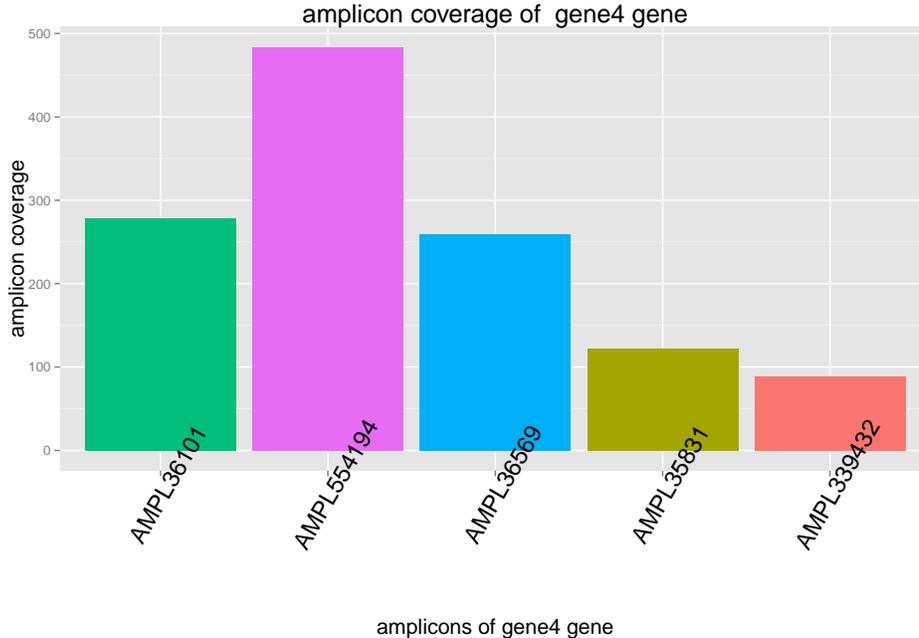


Figure 12: Performance attribute exploration of the *gene4*.

- `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 to 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, in which the x axis represents the genomic position and the y axis, the obtained read counts. The `plotRegion` allows the read profile exploration for a specific genomic region. Helping the region definition, 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, as:

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

Other method, `plotFeature`, allows the read profile exploration of a particular feature. In this case, only is necessary to specify the feature name that should be explored. For example in order to explore the “AMPL20” amplicon of the “gene7”, the code is:

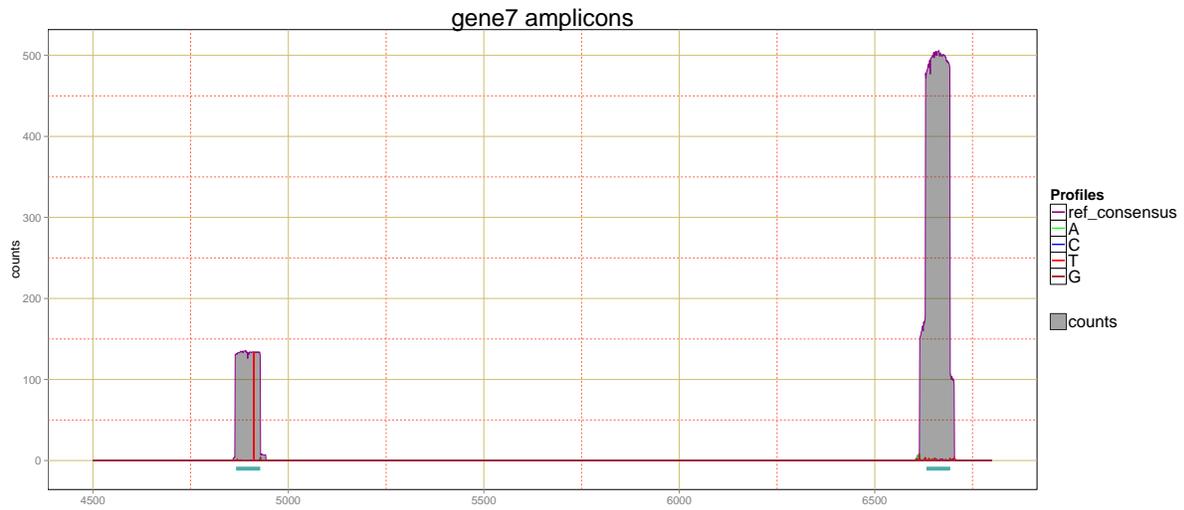


Figure 13: Read counts profile for the gene7 genomic region.

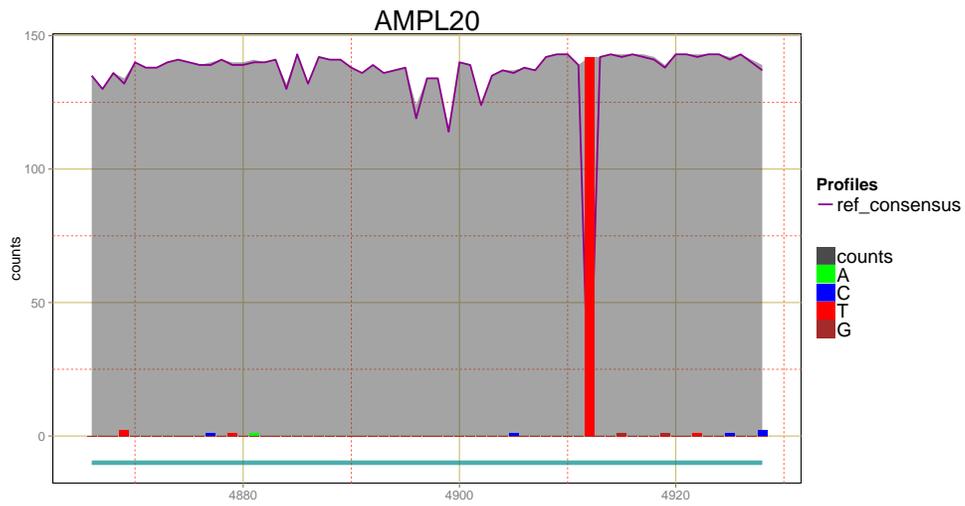


Figure 14: Read counts profile for the "20" gene7 amplicon.

```

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

```

As can be seen in both Figure 13 and Figure 14, the gray shadow corresponds to the total counts that were obtained at each genomic position insight the selected amplicon. The violet line indicates the 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 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 14 can be appreciated that the selected amplicon shows a variation changing the reference nucleotide for a "T". 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. The proportion of read counts that match and no match against the reference can be known using the `plotNtdPercentage` method as:

```

> g<-plotNtdPercentage(myPanel, featureID="AMPL20")
> g

```

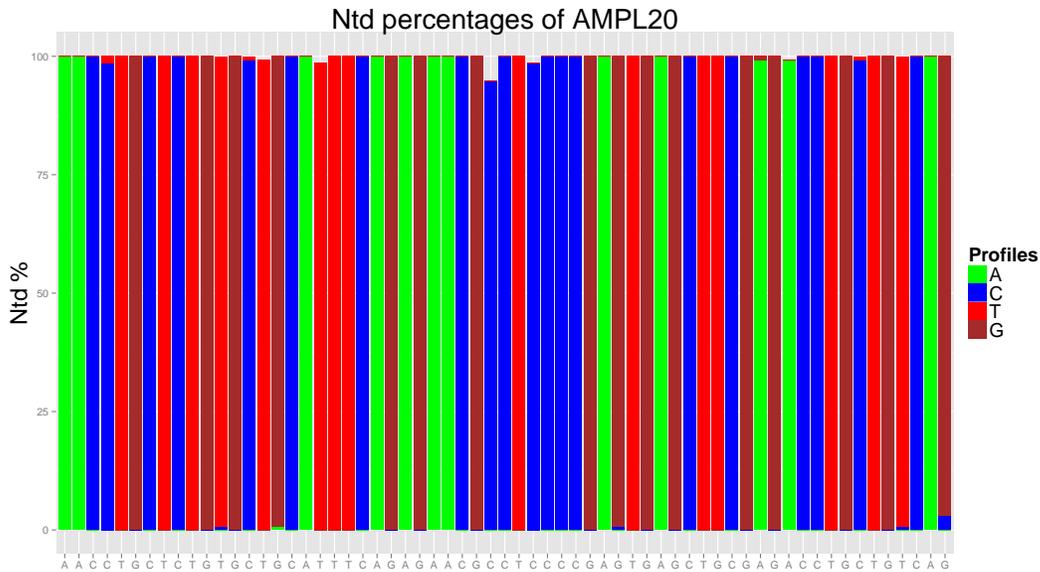


Figure 15: Nucleotide percentages for each genomic position on the "AMPL20" gene7 amplicon.

In Figure 15 can be observed that between 4910 and 4920 positions there are 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
      <Rle>      <IRanges> <Rle> | <character> <numeric> <numeric>
AMPL20  chr10 [4866, 4928]      * |      gene7      0.587      142
      sdCoverage medianCounts IQRCounts
      <numeric>      <numeric> <numeric>
AMPL20          1          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 with 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.

6 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(ampliPanel, attributeThres, imageFile ,file="Results.xlsx")
```

7 Troubleshoot

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
   <Rle>      <IRanges> <Rle> | <character> <numeric> <numeric>
AMPL1      chr1 [ 463,  551]   * |     gene1     0.674     320
AMPL2      chr1 [1553, 1603]   * |     gene2     0.451     550
AMPL3      chr1 [3766, 3814]   * |     gene2     0.531     455
  sdCoverage medianCounts IQRCounts
   <numeric>   <numeric> <numeric>
AMPL1         19         326         24
AMPL2         90         574         14
AMPL3         12         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
   <Rle>      <IRanges> <Rle> |   <numeric> <numeric> <numeric>
gene1      chr1 [ 463,  551]   * |         326         0         320
gene2      chr1 [1553, 3814]   * |         518         56         502
gene3      chr3 [  1,   59]   * |           0           0           0
  sdCoverage
   <numeric>
gene1         0
gene2         67
gene3         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 3.3.0 (2016-05-03)
```

```
Platform: x86_64-pc-linux-gnu (64-bit)
```

```
Running under: Ubuntu 14.04.4 LTS
```

```
locale:
```

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

```
attached base packages:
```

```
[1] stats4    parallel  stats      graphics  grDevices  utils      datasets
[8] methods   base
```

```
other attached packages:
```

```
[1] BiocParallel_1.6.0  TarSeqQC_1.2.0      openxlsx_3.0.0
[4] plyr_1.8.3          ggplot2_2.1.0       Rsamtools_1.24.0
[7] Biostrings_2.40.0  XVector_0.12.0      GenomicRanges_1.24.0
[10] GenomeInfoDb_1.8.0 IRanges_2.6.0       S4Vectors_0.10.0
[13] BiocGenerics_0.18.0
```

```
loaded via a namespace (and not attached):
```

```
[1] Rcpp_0.12.4.5    magrittr_1.5        zlibbioc_1.18.0  munsell_0.4.3
[5] cowplot_0.6.2    colorspace_1.2-6   stringr_1.0.0    tools_3.3.0
[9] grid_3.3.0       gtable_0.2.0       reshape2_1.4.1  bitops_1.0-6
[13] stringi_1.0-1    scales_0.4.0
```

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 technologies—the 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.