TarSeqQC: Targeted Sequencing Experiment Quality Control

Gabriela A Merino[1], Cristóbal Fresno[1], and Elmer A Fernández[1]

[1]CONICET-Universidad Católica de Córdoba, Argentina

March 29, 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 counts intervals can be defined and explored to further assist quality control analysis. Thus, library and pool preparation or sequencing errors could be easily detected. Finally, an .xlsx report containing quality control results can be built.

Contents

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 (**?**). 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 has a large number of interest genomic regions, different PCR pools could be required to achieve a good coverage. All fragments are sequenced in a NGS machine, generating millions of short sequence reads, but its throughput obviously is less than if the whole genome was sequenced. The reads are then aligned against a reference genome and, after that, downstream analysis could be performed. However, prior to further analysis, 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 (**?**).

At present, several open access tools can be used to explore and control experiment results(**?**). 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 are required to be explored and characterized in terms of coverage, as well as, the evaluation and comparison of pool efficiency. 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 amplicon 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 **??** 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*.

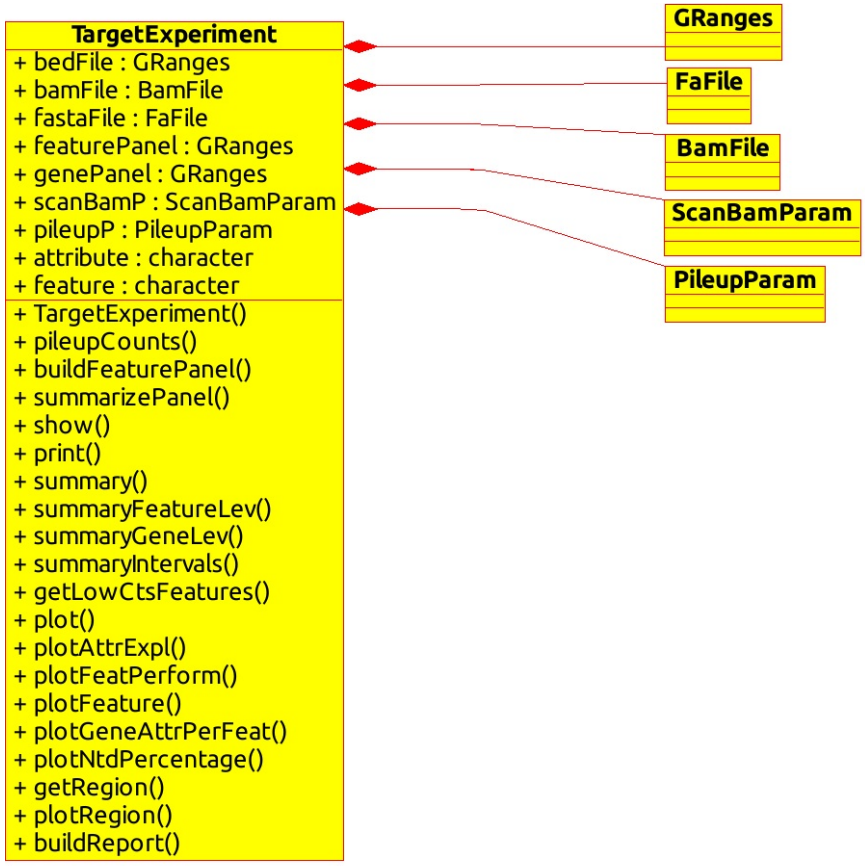

Figure 1: *TargetExperiment* class diagram.

- `fastaFile`: a *FaFile* object that is a reference to the reference sequence.
- `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 illustrate 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 (**?**). 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 to specify the features of interest contained in the *Bed File*, acording to `?` 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 deafult values of their constructors will be used. In addition, `feature` and `attribute` are other important parameters that should be specified in order to conduct the *Quality Control*. 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 meassures will be considered in the *Quality Control* analysis.

3.1 Bed File

The *Bed File* is stored as a *TargetExperiment* slot and is modeled as a *GRanges* object (**?**). The *Bed File* must be a tabular file in which each row represents one sequenced feature. This file should contain at least "chr", "start", "end" , "name" and "gene" 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 neccesary. In this case, the *Bed File* must also contain a "pool" column specifying in which of these pools 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 (**?**). In order to use it to model the *Bed File*, the mandatory fields "chr", "start" and "end" 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 (**?**). In this example case, it corresponds to the amplicon sequencing experiment alignment. This file will be used to build the *pileup* for the selected features in which quality control is based. 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 (**?**) 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 sequences for the selected features. This information is useful to compare the pileup results with the reference, in order to detect *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 (**?**) and this object will be setted as a *TargetExperiment* slot.

3.4 Additional input data

The previous files are mandatories to call the *TargetExperiment* constructor. Additional parameters can be set in order to apply several methods, perform the 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, the same in `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 (**?**)). 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:

```
> 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. This method calls the `pileupCounts` function to build the pileup matrix. Then, the `summarizePanel` is invoked in order to build the `genePanel` slot.
In the previous example, were defined the `feature` and `attribute` parameter values. If you don't do this, you can create the *TargetExperiment* object but a warning message will be printed. Then, you can use the `setFeature` and `setAttribute` methods to set these values. For example:

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

As we mentioned before, when the `scanBamP` and `pileupP` are not specified in the constructor call, they assume their default constructor. But, you could specify those after the constructor call, using `setScanBamP` and `setPileupP`.

```
> # set scanBamP slot value
> scanBamP<-ScanBamParam()
> #set which slot
> bamWhich(scanBamP)<-getBedFile(myPanel)
> setScanBamP(myPanel)<-scanBamP
> # set attribute 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 not be computed. On the other hand, if you do any change in the `scanBamP` and/or `pileupP` slots you will need set the `featurePanel` and the `genePanel` slots again.
The *TarSeqQC* R package provides a dataset that stores the *TargetExperiment* object built with the previous files. In order to use it, you can do:

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

The loaded object is called *ampliPanel*. If you want to use it, you need to re-define the *BAM File* and *FASTA File* path files. In order to do this, you can use:

```
> # 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 neccesary 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 do you call these methods:

```
> # show/print
> myPanel


TargetExperiment
amplicon panel:
        GRanges object with 3 ranges and 5 metadata columns:
        seqnames       ranges strand |        gene medianCounts IQRCounts
           <Rle>    <IRanges>  <Rle> | <character>    <numeric> <numeric>
  AMPL1     chr1 [ 463,  551]      * |       gene1          304        33
  AMPL2     chr1 [1553, 1603]      * |       gene2          560        16
  AMPL3     chr1 [3766, 3814]      * |       gene2          442        36
         coverage sdCoverage
        <numeric>  <numeric>
  AMPL1       297         33
  AMPL2       538         98
  AMPL3       438         26
  -------
  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]      * |          304         0       297
  gene2     chr1 [1553, 3814]      * |          501        59       488
  gene3     chr3 [   1,   59]      * |            0         0         0
        sdCoverage
         <numeric>
  gene1          0
  gene2         71
  gene3          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     247    310  296     372  488
amplicon    0     139    274  299     457  874

> #summary at feature level
> summaryFeatureLev(myPanel)

         Min. 1st Qu. Median Mean 3rd Qu. Max.
amplicon    0     139    274  299     457  874

> #summary at gene level
> summaryGeneLev(myPanel)

     Min. 1st Qu. Median Mean 3rd Qu. Max.
gene    0     247    310  296     372  488
```

Using those methods you can easily find, for example, that in average all amplicons were sequenced at a coverage of 256. You can also see that there is at least one amplicon that was not read. This is because the minimum value of the attribute (*coverage*) is 0. In order to complement this analysis, you could explore the attribute distribution using:

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

In the Figure **??**, the `join` parameter was set as 'TRUE'. 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.

5 Deep exploration and Quality Control

5.1 Panel overview

When you are working with a TS experiment, it is interesting to simultaneously evaluate the performance of all the features. In addition, if you 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 **??**.

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

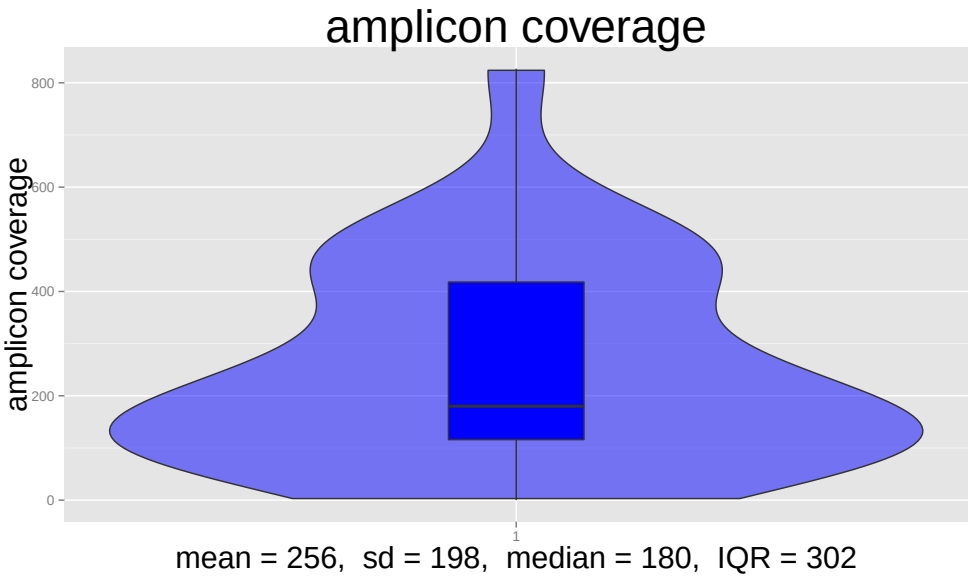

Figure 2: Attribute distribution and density plots.

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*

```
> # 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 it. To help this task, we have implemented the `plot` method. This is a graphical tool consisting in a polar histogram, in which 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.
To build this plot, you can do:

```
> # 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 **??**, the bar corresponding to "gene3" is colored in red and this color is related to the [0,1) coverage interval. In the same plot, we can also appreciate that this gene has

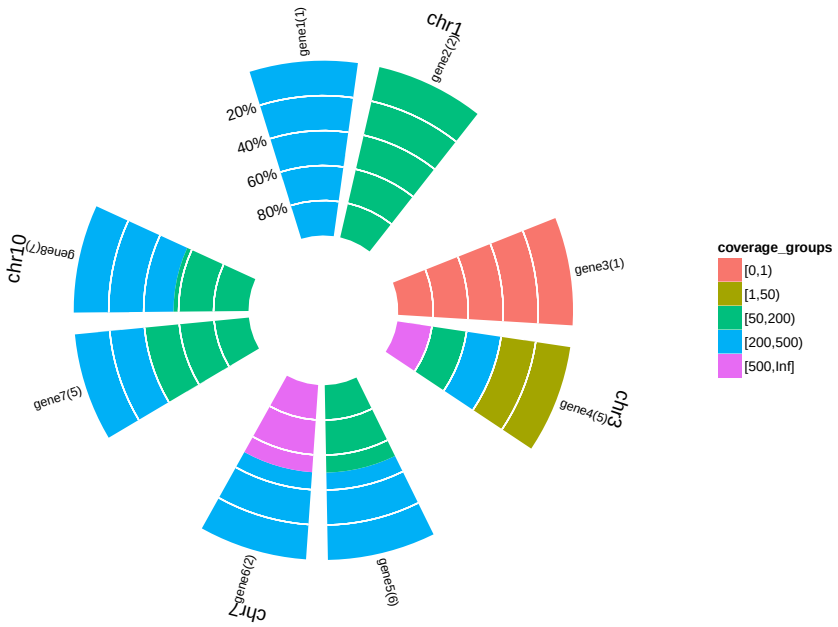

Figure 3: Panel overview plot.

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 a grater impact when you have more features and genes. Figure **??** contains 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* (**?**) that allows to explorae 190 amplicons. In this case, you can easily observe that "MLH1" and "CDKN2A" genes were no sequenced. You can also appreciate that several genes like "ALK", "VHL", "AKT1", "ARBB2", among others, have more uniforme coverage values along their amplicons. On the contrary, "KDR" and "PTEN" genes have some amplicons not sequenced and some other with a high coverage.

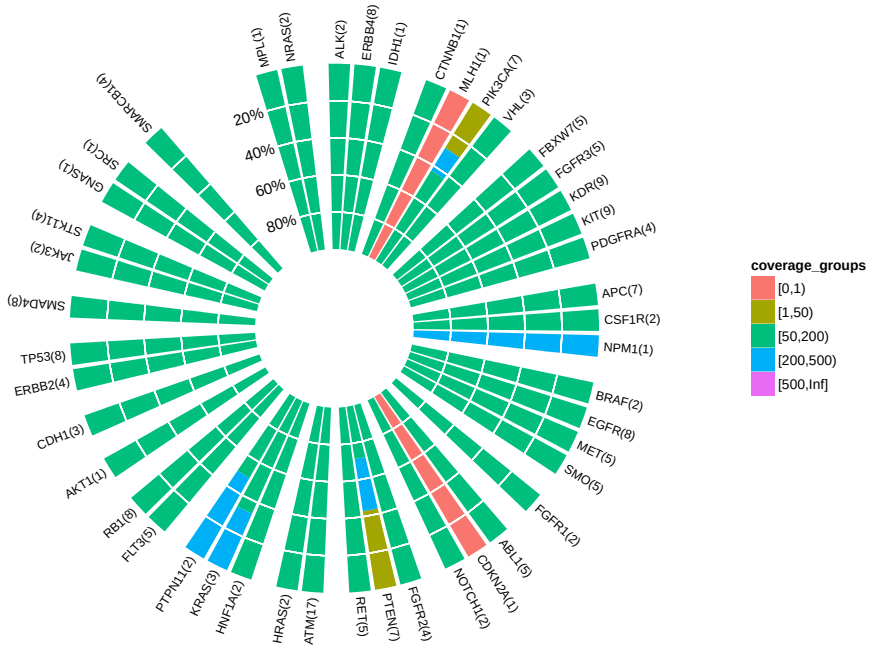

Figure 4: Cancer Panel Primer Pool overview plot.

Complementing the previous plot, `plotFeatPerform` illustrates a similar graphic where the bars are distributed along the x-axis. In order to expand the polar histogram shown in the Figure **??**, 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 mean 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
```

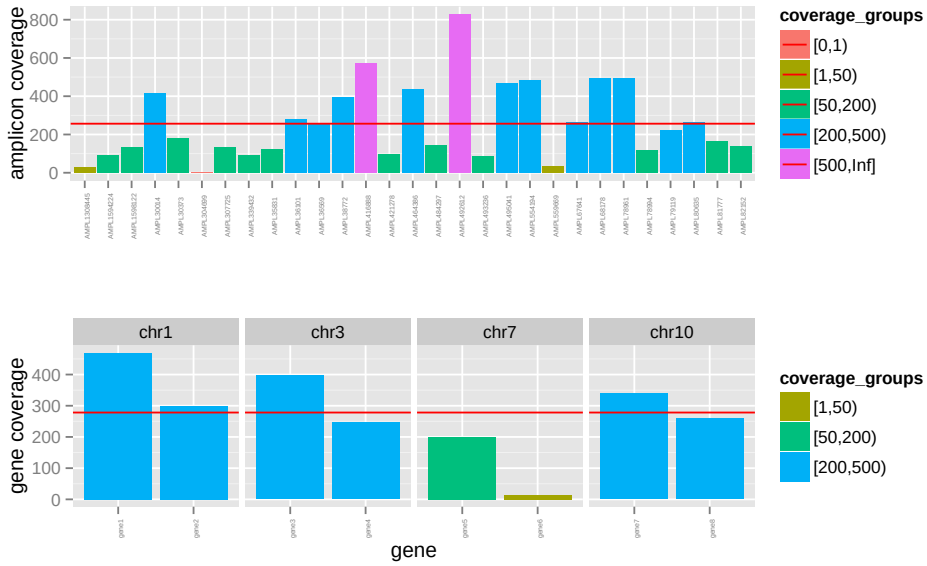

Figure 5: Amplicon coverage performance. The upper panel is a bar plot at feature level, and the lower, at gene level.

In Figure **??** 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 grater 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 Tachnologies* panel that allows the exploration of 16000 amplicons from 409 genes related to several cancer types using 4 primer pools (**?**). In this case, the *Bed File* contains a "pool" column that stores the number pool for each feature. This information will be conserved in the *TargetExperiment* object built from this panel.
In the *Quality Control* context, it is so important to 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 **??** 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 too '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.

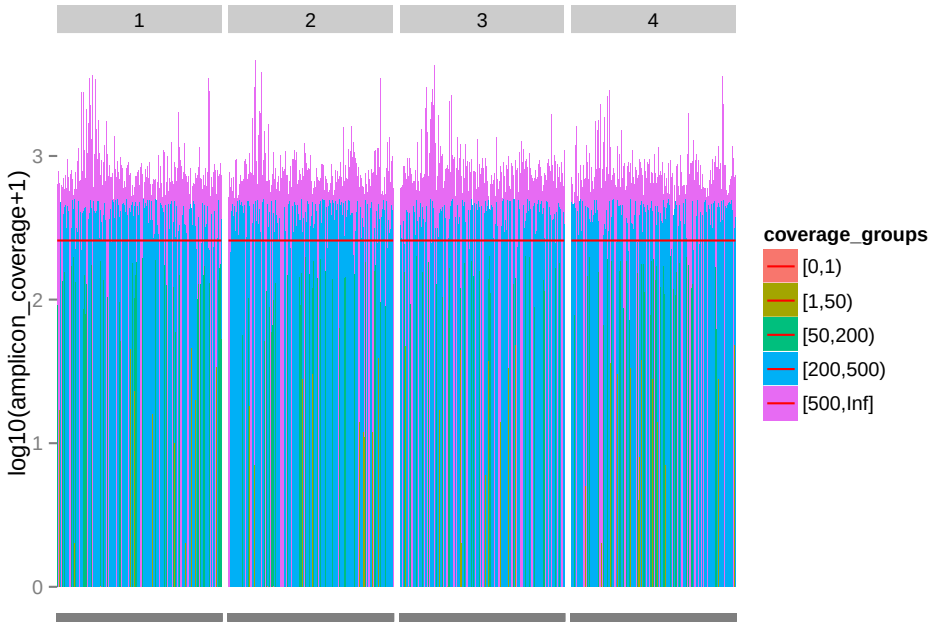

Figure 6: Performance exploration of an Ion AmpliSeq Comprehensive Cancer Panel experiment.

5.2 Controlling low counts features

Low counts features should be detected in early analysis stages. The `summaryIntervals` method builds a frequency table of the fetures 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  12      13 41.4    44.8
4   200  <=  coverage  <  500  10      23 34.5    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 at least it attribute value (*coverage*) lower or higher than a threshold. In this example, you could be interested in knowing how many amplicons have shown at least a coverage of 50, because you consider that this is a minimum value that you will admit. This is a typical aspect that you will explore when you do an experiment *Quality Control*.
Another method that could help you is `getLowCtsFeatures`. This method returns a *data.frame* object that contains all the features that have its attribute value lower than a threshold. The output *data.frame* also contains the panel and attribute information for each feature. For example, if you want 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
1 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 medianCounts IQRCounts coverage sdCoverage
1 AMPL4    chr3     1  59 gene3            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
```

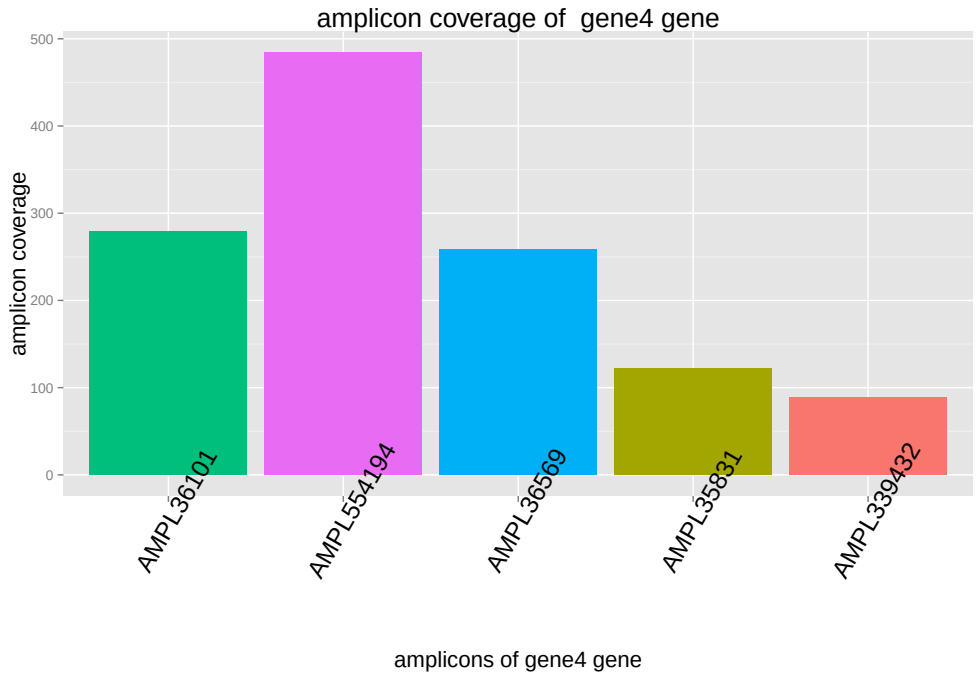

Figure 7: Performance attribute exploration of the *gene4*.

In Figure **??** you can observe the attribute value for each feature contained in the "gene4" gene.

5.3 Read counts exploration

When you are doing a *Quality Control* the analysis of coverage/median counts achieved for each feature is important. But, sometimes could be interesting in exploring the read profile obtained for a particular genomic region or a feature. For this reason, the *TarSeqQC* R package provides methods to help the exploration at a nucleotide resolution.
Remember that when the `featurePanel` slot of a *TargetExperiment* object is built, the `pileupCounts` function is called. This and the `buildFeaturePanel` method are responsible for the pileup construction, read counts obtainment, and *coverage/medianCounts* calculation. Even though these functions are internally invoked by the *TargetExperiment* constructor, if you wish, you can call them. In particular, when `pileupCounts` is invoked, you will obtain a data.frame that contains the read counts information for each position that was specified in the *Bed File*. In addition, this is a function, not a *TargetExperiment* method that could be called externally to the class. Note that the columns in the obtained object 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.
In order to call the `pileupCounts` function is neccesary to specify several parameters:

- `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 a *BamFile* object. If it was not specified, 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*. If it was not specified, the `pileupP` parameter will be defined using the constructor default values.

In our case, to work with the example data, you could 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   0   3   2 289 0 289 1 chr1:463-551    295
1346 464     chr1   A 297   0   1   1 0 297 1 chr1:463-551    300
1347 465     chr1   G   1   0 305   0 0 305 0 chr1:463-551    306
1348 466     chr1   T   0   0   1 306 0 306 0 chr1:463-551    307
1349 467     chr1   G   0   0 311   0 0 311 0 chr1:463-551    311
1350 468     chr1   C   0 316   0   0 0 316 0 chr1:463-551    316
```

Using the obtained read count information it is possible to build a *read profile* plot, in which the x axis represents the genomic position and the y axis, the obtained read counts. It is also important to distinguish how many read counts correspond to the reference nucleotide and how many could correspond to a genomic variation. 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:

```
> #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, you could use the previous information to specify a genomic region, as:

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

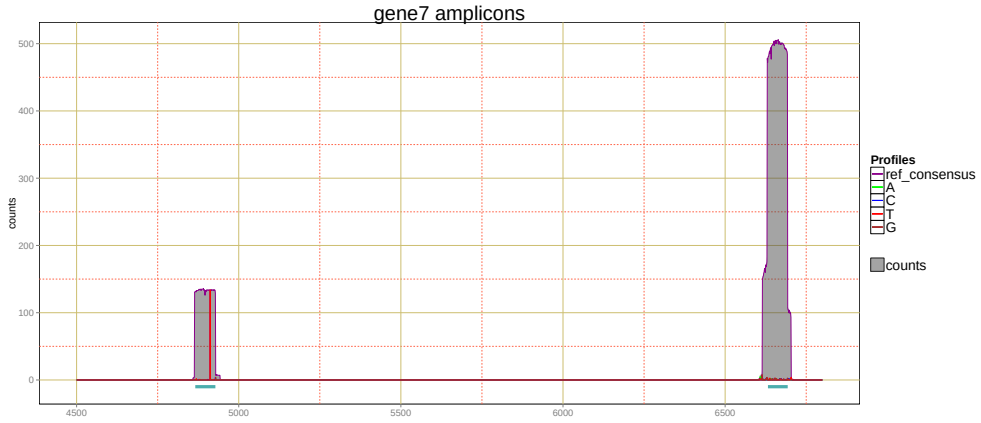

Figure 8: Read counts profile for the gene7 genomic region.

The `plotFeature` allows the read profile exploration of a particular feature. For example if we wish to explore the "AMPL20" amplicon of the "gene7", we should do:

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

As you can see in Figure **??**, the gray shadow correspond 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. The green, blue, red and brown lines illustrate the read counts that do not match with the reference and inform about the detected nucleotide variation. In this example, the selected amplicon show a variation that change the reference nucleotide for a "T". If you wan to know exactly the proportion of read counts that match and no match against the reference, you can use the `plotNtdPercentage` as:

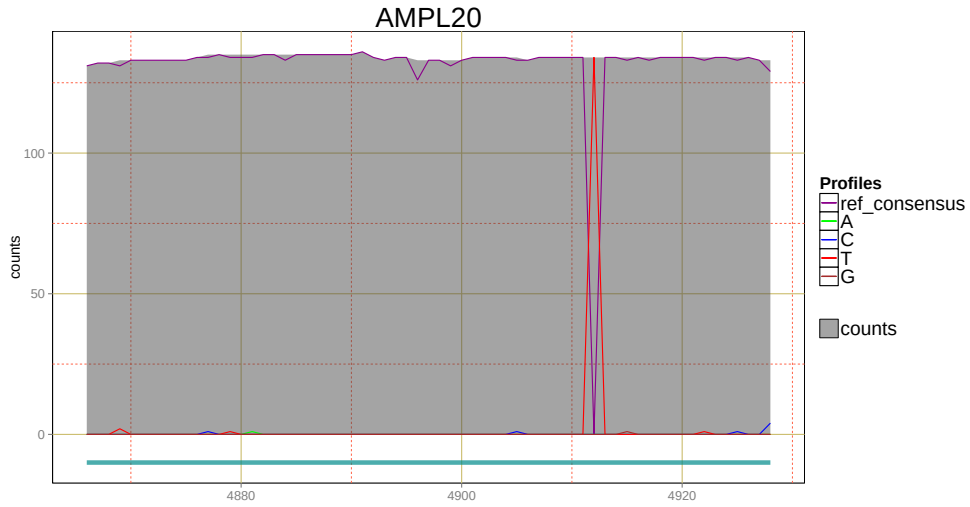

Figure 9: Read counts profile for the "20" gene7 amplicon.

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

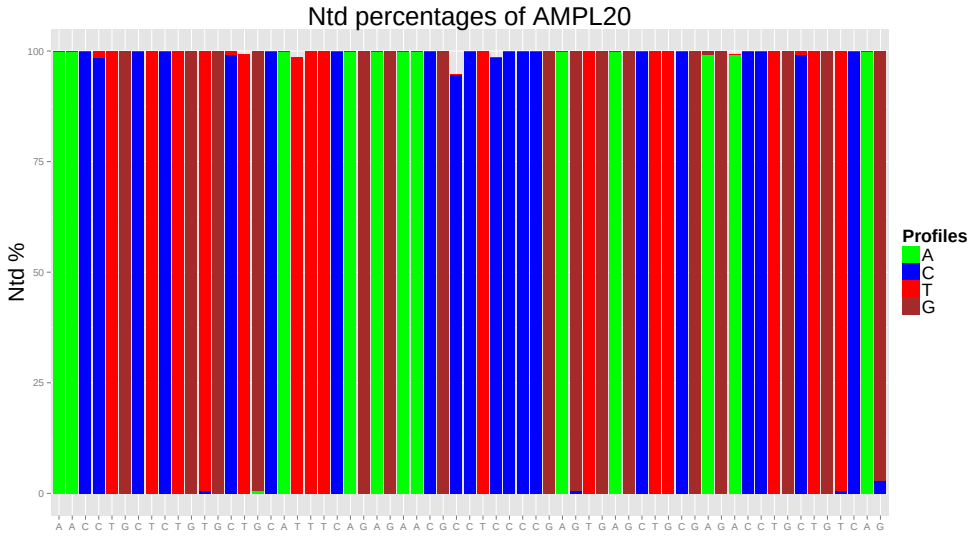

Figure 10: Nucleotide percentages for each genomic position on the "AMPL20" gene7 amplicon.

In Figure **??** you can observe that in the position 4912 of the reference genome indicates that there should be a "G" and the read counts indicate that in this position is a "T". You could also extract this information using the previous read counts *data.frame* `myCounts`. For this, remember that the `featurePanel` slot is a *GRanges* object. Then, you could subset this directly using the feature name:

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

```
GRanges object with 1 range and 5 metadata columns:
         seqnames       ranges strand |        gene medianCounts IQRCounts
            <Rle>    <IRanges>  <Rle> | <character>    <numeric> <numeric>
  AMPL20    chr10 [4866, 4928]      * |       gene7          140         5
          coverage sdCoverage
         <numeric>  <numeric>
  AMPL20       139          5
  -------
  seqinfo: 4 sequences from an unspecified genome; no seqlengths
```

Using this information, you could subset in `myCounts` object only those rows that corresponding with the feature:

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

Then, you could find which position have the lowest value in the "=" column. It means, the minimum value of read counts matching against the reference:

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

```
      pos seqnames seq A C G   T N = -     which_label counts
1423 4912    chr10   G 0 0 0 142 0 0 0 chr10:4866-4928    142
```

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 that characterize the experiment. You could choose any graphic, but if you don not specify its name, the method calls the `plot` *TarSeqQC* 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 you only want to generate the report, you could call the `buildReport` after the object construction. In our case, we want to specify the image file that we want to include in the report, to do this, we should do:

```
> 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, if you use some of them, please make

sure that the corresponding *TargetExperiment* slots have the file names well defined. For example, if you wish loading the *TarSeqQC* example data, you can do:

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

```
TargetExperiment
amplicon panel:
        GRanges object with 3 ranges and 5 metadata columns:
        seqnames       ranges strand |        gene medianCounts IQRCounts
           <Rle>    <IRanges>  <Rle> | <character>    <numeric> <numeric>
  AMPL1     chr1 [ 463,  551]      * |       gene1          182        14
  AMPL2     chr1 [1553, 1603]      * |       gene2          493        14
  AMPL3     chr1 [3766, 3814]      * |       gene2          423        22
         coverage sdCoverage
        <numeric>  <numeric>
  AMPL1       180         16
  AMPL2       470         83
  AMPL3       418         11
  -------
  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]      * |          182         0       180
  gene2     chr1 [1553, 3814]      * |          458        35       444
  gene3     chr3 [   1,   59]      * |            0         0         0
        sdCoverage
         <numeric>
  gene1          0
  gene2         37
  gene3          0
  -------
  seqinfo: 4 sequences from an unspecified genome; no seqlengths

selected attribute:
         coverage
```

But, if you want to re-built the `featurePanel` slot, the `pileupCounts` execution will cause an error because the method cannot find the files.

```
buildFeaturePanel(ampliPanel)
[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, you should do:

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

and then:

```
> setFeaturePanel<-buildFeaturePanel(ampliPanel)
```

Session Info

```
> sessionInfo()


R version 3.2.4 (2016-03-10)
Platform: x86_64-apple-darwin13.4.0 (64-bit)
Running under: OS X 10.9.5 (Mavericks)

locale:
[1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

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

other attached packages:
 [1] BiocParallel_1.4.3   TarSeqQC_1.0.3       openxlsx_3.0.0
 [4] plyr_1.8.3           ggplot2_2.1.0        Rsamtools_1.22.0
 [7] Biostrings_2.38.4    XVector_0.10.0       GenomicRanges_1.22.4
[10] GenomeInfoDb_1.6.3   IRanges_2.4.8        S4Vectors_0.8.11
[13] BiocGenerics_0.16.1

loaded via a namespace (and not attached):
 [1] Rcpp_0.12.4          magrittr_1.5         zlibbioc_1.16.0
 [4] cowplot_0.6.1        munsell_0.4.3        colorspace_1.2-6
 [7] stringr_1.0.0        tools_3.2.4          grid_3.2.4
[10] gtable_0.2.0         lambda.r_1.1.7       futile.logger_1.4.1
[13] reshape2_1.4.1       futile.options_1.0.0 bitops_1.0-6
[16] stringi_1.0-1        scales_0.4.0
```