BadRegionFinder – an R/Bioconductor package for identifying regions with bad coverage

Sarah Sandmann

May 19, 2021

Contents

1 Introduction

In the use case of targeted sequencing, determining the coverage of the targeted regions is a crucial step. A sufficient coverage of the target regions is essential. Otherwise, all subsequent experiments might fail. Mutations might for example not be detected, because the corresponding regions are hardly covered.

The reasons for observing bad coverage are diverse. Not enough material might have been used for sequencing. Or a random error might have occurred during sequencing itself. Yet, these

problems do usually result in a continuously bad coverage of a single sample, resp. those samples that were analyzed mutually. A different type of problem results from genes or regions in the genome, which are generally difficult to sequence. These difficulties often result from a high GC-content [4]. Another problem might be primers, which were not designed optimally. If the two primers of an amplicon are located too far away from each other, a decrease in coverage is expected to be observed for all samples concerning parts of this amplicon.

Plenty of programs exist for performing a coverage analysis. Programs like BEDtools [3] may quickly calculate the coverage of any given bam file. Thereby, the coverage of any defined windows or per base may be calculated. Similarly, the R package 'bamsignals' [2] is able to extract information on the coverage of every position of interest from a given bam file.

When designing a targeted sequencing experiment and it comes to identifying regions, which show a generally bad coverage, it is necessary to combine the coverage of many samples. It appears useful to have a tool, that automatically combines the coverage information of different samples and provides different possibilities for analysis.

BadRegionFinder is a package that allows for a coverage analysis of various samples at a time. The user is able to define individual thresholds that divide the coverage into three categories: bad, acceptable and good coverage. The whole genome may be considered as well as a set of target regions. Thereby, *BadRegionFinder* does not only provide a possibility for identifying regions with a bad coverage, but also regions with a high coverage off target indicating that a pair of the designed primers might not be mapping uniquely or that there might be problems in the alignment.

For the final analysis of the coverage quality different options are available:

- summary variant, focusing on regions
- detailed variant (basewise results)
- summary variant, focusing on genes

1.1 Loading the package

The package can be downloaded and installed with

```
> BiocManager::install("BadRegionFinder")
```

After installation, the package can be loaded into R by typing

```
> library(BadRegionFinder)
```

into the R console.

BadRegionFinder requires the R-packages *VariantAnnotation*, *Rsamtools*, *biomaRt*, *GenomicRanges* and *S4Vectors*, as well as functions from the R-packages *utils*, *stats*, *grDevices* and *graphics*. All of them are loaded automatically when using *BadRegionFinder*. Furthermore the package *BSgenome.Hsapiens.UCSC.hg19* is suggested.

2 Determining and classifying the coverage

Prior to reporting and visualizing information on the coverage over all samples, the coverage of every sample at every position of interest has to be determined, the information has to be combined and classes concerning the coverage quality have to be assigned. To do so, different functions are available.

2.1 Determine the coverage

If the user is interested in investigating the coverage on and off target considering the whole genome, it is recommended to use the function `determineCoverage` with `TRonly = FALSE`. The function scans the whole genome and wherever a base is covered by at least one read or an originally targeted base is detected, detailed information concerning this position is written out.

If the user is interested in investigating the coverage on target only, it is recommended to use the function `determineCoverage` with `TRonly = TRUE`. Again, the whole genome gets scanned, but only wherever a targeted base is registered, detailed information concerning this position is written out.

For the correct functioning of `determineCoverage`, various input objects are necessary:

- The names of the samples to be analyzed have to be provided by a data frame object. There has to be one sample name per line without the ".bam" suffix.

Exemplary data frame object *samples*:

```
> sample_file <- system.file("extdata", "SampleNames2.txt", package = "BadRegionFinder")
> samples <- read.table(sample_file)
> samples

         V1
1 Test2_454
2 Test2_ion
```

- The bam- and the corresponding bai files of the samples to be analyzed have to be provided in a folder. The names of the files have to match the names provided by the sample names file. Alternatively, a bamFileList may be defined.
- The target regions have to be provided by a data frame- or a GRanges object. If a data frame object is provided, the name of the chromosome (without "chr") has to be defined in the first column, the start position of the target region has to be defined in the second column and the end position of the region has to be defined in the third column.

Exemplary data frame object *targetRegions*:

```
> target_regions <- system.file("extdata", "targetRegions2.bed", package = "BadRegionFinder")
> targetRegions <- read.table(target_regions, header = FALSE, stringsAsFactors = FALSE)
> targetRegions
```

```
  V1        V2        V3
1  2 198266410 198267032
2  2 198267198 198267820
3 17  74732937  74733301
```

- The folder where the output shall be saved may be defined as well. For every chromosome that is defined in the genome, a file is written out: "Summary_chr<chromosomename>.txt". Apart from information on the coverage, this file also contains information on whether the analyzed bases are part of the target region (targetBases = 1) or not (targetBases = 0). If only an empty string is provided, no files are written out. The results are stored in a GRangesList object.

Exemplary results for chromosome 2 (`TRonly = FALSE`):

```
> bam_input <- system.file("extdata", package = "BadRegionFinder")
> coverage_summary <- determineCoverage(samples, bam_input, targetRegions, "", TRonly = FALSE)
> coverage_summary[[2]]

GRanges object with 1611 ranges and 3 metadata columns:
         seqnames              ranges strand | Test2_454 Test2_ion targetBases
            <Rle>           <IRanges>  <Rle> | <numeric> <numeric>   <numeric>
     [1]        2         1-198266311      * |         0         0           0
     [2]        2           198266312      * |        48       128           0
     [3]        2           198266313      * |        48       128           0
     [4]        2           198266314      * |        48       128           0
     [5]        2           198266315      * |        48       128           0
     ...      ...                 ...    ... .       ...       ...         ...
  [1607]        2           198267917      * |         1       119           0
  [1608]        2           198267918      * |         1       119           0
  [1609]        2           198267919      * |         1       119           0
  [1610]        2           198267920      * |         1       119           0
  [1611]        2 198267921-243199373      * |         0         0           0
  -------
  seqinfo: 25 sequences from an unspecified genome; no seqlengths
```

- Furthermore, it has to be defined, whether the coverage concerning the whole genome shall be reported (`TRonly = FALSE`), or only the coverage concerning the target regions (`TRonly = TRUE`).

2.2 Determine the coverage quality

Apart from determining the coverage of every sample at all the positions of interest, it is essential to combine this information and to classify the coverage quality at every position. This is done by the function `determineCoverageQuality`. The function analyzes every position in the return value of `determineCoverage` (with `TRonly = FALSE` or `TRonly = TRUE`) and determines the corresponding coverage quality.

There exist six different categories of coverage quality: bad coverage off target, bad coverage on target, acceptable coverage off target, acceptable coverage on target, good coverage off target,

good coverage on target. These categories are numerically coded and user-defined. To define the thresholds, four essential parameters that have to be set:

- threshold1: the first coverage threshold
- percentage1: the first percentage threshold
- threshold2: the second coverage threshold
- percentage2: the second percentage threshold

To categorize the coverage quality, the coverage of each sample is first of all categorized according to `threshold1` and `threshold2` into three different categories:

- bad coverage: less than `threshold1` reads
- acceptable coverage: at least `threshold1`, but less than `threshold2` reads
- good coverage: at least `threshold2` reads

Subsequently this information gets combined with the defined precentages to obtain a numerically coded quality value for each region saved in the previously created GRangesList object `coverage_summary`:

- 0: off target; not even `percentage1` percent of all samples have a good or acceptable coverage (bad region)
- 1: on target; not even `percentage1` percent of all samples have a good or acceptable coverage (bad region)
- 2: off target; at least `percentage1` percent of all samples have a good or acceptable coverage, but less than `percentage2` percent of all samples have a good coverage (acceptable region)
- 3: on target; at least `percentage1` percent of all samples have a good or acceptable coverage, but less than `percentage2` percent of all samples have a good coverage (acceptable region)
- 4: off target; at least `percentage2` percent of all samples have a good coverage (good region)
- 5: on target; at least `percentage2` percent of all samples have a good coverage (good region)

In addition to `threshold1`, `percentage1`, `threshold2` and `percentage2` the GRangesList object, that is the return value of `determineCoverage`, is necessary for the correct functioning of `determineCoverageQuality`.

As the output is still in a very raw version, it is not directly saved by the function, but returned as a list object.

Exemplary output object *coverage_indicators* (chromosome 2):

```
> threshold1 <- 20
> threshold2 <- 100
> percentage1 <- 0.80
> percentage2 <- 0.90
> coverage_indicators <- determineCoverageQuality(threshold1, threshold2, percentage1,
+                                                 percentage2, coverage_summary)
> coverage_indicators[[2]]
```

```
GRanges object with 1611 ranges and 4 metadata columns:
         seqnames              ranges strand | Test2_454 Test2_ion targetBases indicator
            <Rle>           <IRanges>  <Rle> | <numeric> <numeric>   <numeric> <numeric>
     [1]        2         1-198266311      * |         0         0           0         0
     [2]        2           198266312      * |        48       128           0         2
     [3]        2           198266313      * |        48       128           0         2
     [4]        2           198266314      * |        48       128           0         2
     [5]        2           198266315      * |        48       128           0         2
     ...      ...                 ...    ... .       ...       ...         ...       ...
  [1607]        2           198267917      * |         1       119           0         0
  [1608]        2           198267918      * |         1       119           0         0
  [1609]        2           198267919      * |         1       119           0         0
  [1610]        2           198267920      * |         1       119           0         0
  [1611]        2 198267921-243199373      * |         0         0           0         0
  -------
  seqinfo: 25 sequences from an unspecified genome; no seqlengths
```

2.3 Determine regions of interest

If the user is not interested in an analysis of the whole genome or of the whole target region, but just in a selction of regions (on and/or off target), these regions may be selected by the help of the function `determineRegionsOfInterest`. For every base defined to be in a *region of interest*, the previously determined coverage information is written out. This information always includes the coverage of each sample and it may include the assigned class of coverage quality - depending on the input object.

For the correct functioning of `determineRegionsOfInterest` two input objects are necessary:

- The regions of interest have to be provided by a data frame- or GRanges object. In case of a data frame object, the name of the chromosome (without "chr") has to be defined in the first column, the start position of the region of interest has to be defined in the second column and the end position of the region of interest has to be defined in the third column.

Exemplary data frame object *regionsOfInterest*:

```
> regionsOfInterest<-data.frame(chr = c(2,2,17),
+                               start = c(198266420,198267200,74732940),
+                               end = c(198267032,198267800,74733301))
> regionsOfInterest

  chr     start       end
1   2 198266420 198267032
2   2 198267200 198267800
3  17  74732940  74733301
```

- The input, which is the return value of `determineCoverageQuality` or `determineCoverage` (with `TRonly = FALSE` or `TRonly = TRUE`) (or `determineRegionsOfInterest`), has to be provided by a list object. Every component of the list considers one chromosome.

Exemplary output object *coverage_indicators_2* (chromosome 2):

```
> coverage_indicators_2 <- determineRegionsOfInterest(regionsOfInterest, coverage_indicators)
> coverage_indicators_2[[2]]

GRanges object with 1214 ranges and 4 metadata columns:
         seqnames    ranges strand | Test2_454 Test2_ion targetBases indicator
            <Rle> <IRanges>  <Rle> | <numeric> <numeric>   <numeric> <numeric>
     [1]        2 198266420      * |        85       179           1         3
     [2]        2 198266421      * |        85       179           1         3
     [3]        2 198266422      * |        85       179           1         3
     [4]        2 198266423      * |        85       179           1         3
     [5]        2 198266424      * |        85       179           1         3
     ...      ...       ...    ... .       ...       ...         ...       ...
  [1210]        2 198267796      * |       526       304           1         5
  [1211]        2 198267797      * |       526       304           1         5
  [1212]        2 198267798      * |       526       303           1         5
  [1213]        2 198267799      * |       526       301           1         5
  [1214]        2 198267800      * |       526       301           1         5
  -------
  seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

It is not recommended to use `determineCoverage` with `TRonly=TRUE`, but to select regions off target using `determineRegionsOfInterest`. In this case, no coverage could be registered for all bases off target, as this information was not saved during the step of `determineCoverage`.

If `regionsOfInterest` is identical compared to `targetRegions` and the function `determineCoverage` with `TRonly=TRUE` was applied in the first case, the selection of regions that is returned is not changed at all.

3 Reporting the results

The function `determineCoverage` already produces some output files. Yet, these files just contain raw information on the coverage of every sample at every position in the whole genome, resp. the target region. Information on the coverage quality is not yet included in these files. To report this information, three different variants are available.

3.1 Summary variant (regions)

If the user likes to gain an overview of the coverage quality, it is recommended to use the function `reportBadRegionsSummary` to create a summary report considering all regions of interest, their coverage quality and the corresponding gene (including gene name and geneID).

The function scans every position in the input object. Wherever subsequent bases feature the same coverage quality, the region gets summed up. Although it is not directly reported whether a region contains on or off target bases, this information can be gained from the coverage quality: all bases off target feature an even number characterizing the coverage quality; all bases on target feature an uneven number characterizing the coverage quality.

For every summed up region the gene that is most likely to be targeted by the original experiment gets reported using biomaRt. If no gene can be found, "NA" is saved for the corresponding region. If not all bases in the summed up region cover a gene, the gene gets reported for the whole region nonetheless.

For the correct functioning of `reportBadRegionsSummary` various input objects are necessary:

- Values defining the classes of coverage (`threshold1`, `percentage1`, `threshold2`, `percentage2`; for more details on the classes see section 2.2).
- The input, which is the return value of `determineCoverageQuality` or `determineRegionsOfInterest`, has to be provided by a list object. Every component of the list considers one chromosome.
- If the information concerning the genes is not supposed to result from the human genome (hg19), a mart has to be provided (compare R/Bioconductor package biomaRt [1]). Otherwise, the human genome (hg19) is used by default if an empty string is provided.

Exemplary mart (default mart):

```
> library(biomaRt)
> mart = useMart(biomart="ENSEMBL_MART_ENSEMBL",host="grch37.ensembl.org",
+                path="/biomart/martservice",dataset="hsapiens_gene_ensembl")
> mart

Object of class 'Mart':
  Using the ENSEMBL_MART_ENSEMBL BioMart database
  Using the hsapiens_gene_ensembl dataset
```

- The folder where the output shall be saved may be defined. A single a file is written out: "BadCoverageSummary`threshold1`;`percentage1`;`threshold2`;`percentage2`.txt". If no output shall be written out, an empty string may be passed.

Exemplary return value:

```
> badCoverageSummary <- reportBadRegionsSummary(threshold1, threshold2, percentage1,
+                                               percentage2, coverage_indicators_2, mart, "")
> badCoverageSummary

GRanges object with 7 ranges and 3 metadata columns:
      seqnames              ranges strand | QualityMarker        Gene          GeneID
         <Rle>           <IRanges>  <Rle> |     <numeric> <character>     <character>
  [1]        2 198266420-198266611      * |             3       SF3B1 ENSG00000115524
  [2]        2 198266612-198266725      * |             5       SF3B1 ENSG00000115524
  [3]        2 198266726-198266811      * |             3       SF3B1 ENSG00000115524
  [4]        2 198266812-198267032      * |             1       SF3B1 ENSG00000115524
  [5]        2 198267200-198267256      * |             3       SF3B1 ENSG00000115524
  [6]        2 198267257-198267800      * |             5       SF3B1 ENSG00000115524
  [7]       17   74732940-74733301      * |             1       SRSF2 ENSG00000161547
  -------
  seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

The output file may be visualized using the function `plotSummary` (see section 4.1).

3.2 Detailed variant

If the user likes to receive more detailed information on the coverage quality, it is recommended to use the function `reportBadRegionsDetailed` to create a detailed report considering all regions of interest (basewise), the coverage of each sample at the corresponding positions, the indicator whether the bases were originally targeted, their coverage quality and the corresponding gene (name and geneID).

Different from the summed-up variant `reportBadRegionsSummary`, information on every single base of interest gets reported (except for completely uncovered and untargeted regions, which are excluded). For every base its position, the coverage of each sample, information on whether this base was originally targeted (value 1) or not (value 0), the coverage quality and the most likely gene (name and geneID) that was targeted by the original experiment get reported. Information on the gene names and the geneIDs result from biomaRt [1]. If no gene can be found at a position, "NA" is reported for the corresponding base.

For the correct functioning of `reportBadRegionsSummary` various input objects are necessary:

- Values defining the classes of coverage (`threshold1`, `percentage1`, `threshold2`, `percentage2`; for more details on the classes see section 2.2).
- The input, which is the return value of `determineCoverageQuality` or `determineRegionsOfInterest`, has to be provided by a list object. Every component of the list considers one chromosome.
- If the information concerning the genes is not supposed to result from the human genome (hg19), a mart has to be provided (compare R/Bioconductor package biomaRt [1]). Otherwise, the human genome (hg19) is used by default if an empty string is provided.
- The names of the samples that were analyzed (just used for naming the columns in the output file correctly) have to be provided by a data frame object. There has to be one sample name per line. The folder where the output shall be saved may be defined. A single a file is written out: "BadCoverageChromosome<chromosomename>;`threshold1`;`percentage1`;`threshold2`;`percentage2`.txt". If no output shall be written out, an empty string may be passed.

Exemplary return value for chromosome 2:

```
> coverage_indicators_temp <- reportBadRegionsDetailed(threshold1, threshold2, percentage1,
+                                                      percentage2, coverage_indicators_2, "",
+                                                      samples, "")
> coverage_indicators_temp[[2]]

GRanges object with 1214 ranges and 6 metadata columns:
         seqnames    ranges strand | Test2_454 Test2_ion targetBases QualityMarker        Gene
            <Rle> <IRanges>  <Rle> | <numeric> <numeric>   <numeric>     <numeric> <character>
     [1]        2 198266420      * |        85       179           1             3       SF3B1
     [2]        2 198266421      * |        85       179           1             3       SF3B1
     [3]        2 198266422      * |        85       179           1             3       SF3B1
     [4]        2 198266423      * |        85       179           1             3       SF3B1
     [5]        2 198266424      * |        85       179           1             3       SF3B1
     ...      ...       ...    ... .       ...       ...         ...           ...         ...
```

```
[1210]        2 198267796      * |       526       304           1             5       SF3B1
[1211]        2 198267797      * |       526       304           1             5       SF3B1
[1212]        2 198267798      * |       526       303           1             5       SF3B1
[1213]        2 198267799      * |       526       301           1             5       SF3B1
[1214]        2 198267800      * |       526       301           1             5       SF3B1
                GeneID
            <character>
   [1] ENSG00000115524
   [2] ENSG00000115524
   [3] ENSG00000115524
   [4] ENSG00000115524
   [5] ENSG00000115524
   ...             ...
[1210] ENSG00000115524
[1211] ENSG00000115524
[1212] ENSG00000115524
[1213] ENSG00000115524
[1214] ENSG00000115524
-------
seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

The output file may be visualized using the function `plotDetailed` (see section 4.2).

3.3 Summary variant (genes)

If the user likes to gain an overview of the coverage quality of each targeted gene, use of the function `reportBadRegionsGenes` is recommended. The function creates a summary report considering the coverage quality on a genewise level considering all regions of interest.

`reportBadRegionsGenes` sums up all regions covering the same gene in the following way: The number of bases falling into each quality category is summed up. Thereby, regions which were orignially targeted may easily be separated from those which were not, as targeted regions always feature an uneven number characterizing their coverage quality. If a region is broader than the detected gene, but the quality category is the same for the whole region, the whole region is assigned to the gene. If no gene is reported in the input file, the coverage quality is summed up for a gene named "NA". In a final step, the absolute number of bases is converted to a relative figure to allow for a better comparison between the different genes.

For the correct functioning of `reportBadRegionsGenes` various input objects are necessary:

- Values defining the classes of coverage (`threshold1`, `percentage1`, `threshold2`, `percentage2`; for more details on the classes see section 2.2).
- The input, which is the return value of `reportBadRegionsSummary` has to be provided (data frame object).
- The folder where the output shall be saved may be defined. A single file is written out: "BadCoverageGenes`threshold1`;`percentage1`;`threshold2`;`percentage2`.txt". If no output shall be written out, an empty string may be passed.

Exemplary return value:

```
> badCoverageOverview <- reportBadRegionsGenes(threshold1, threshold2, percentage1, percentage2,
+                                              badCoverageSummary, "")
> badCoverageOverview

   Gene          GeneID BadRegion_offTarget BadRegion_onTarget AcceptableRegion_offTarget
1 SF3B1 ENSG00000115524                   0          0.1820428                           0
2 SRSF2 ENSG00000161547                   0          1.0000000                           0
  AcceptableRegion_onTarget GoodRegions_offTarget GoodRegions_onTarget
1                 0.2759473                     0            0.5420099
2                 0.0000000                     0            0.0000000
```

The output file may be visualized using the function `plotSummaryGenes` (see section 4.3).

4 Visualizing the results

Apart from creating various textual reports, summing up the results of the coverage analysis, there exist functions to automatically visualize each type of report.

4.1 Summary variant (regions)

If the user likes to visualize the output of `reportBadRegionsSummary`, it is recommended to use the function `plotSummary`. A line graph is returned, visualizing the number of bases that fall into each category of coverage quality. Furthermore, information on the genes located in these regions is included:

On the y axis the coverage quality is coded. The different categories are color coded as well as height coded. As numbers from 0 to 5 were previously assigned to the different categories, thick lines are now drawn at the height of the category. Furthermore, The categories are color coded in the following way: red - bad region on target; yellow - acceptable region on target; green - good region on target; black - bad region off target; dark gray - acceptable region off target; light gray - good region off target.

On the x axis the detected genes are printed. Wherever a new region covering a new gene is registered, a dashed line is drawn.

For the correct functioning of `reportBadRegionsSummary` various input objects are necessary:

- Values defining the classes of coverage (`threshold1`, `percentage1`, `threshold2`, `percentage2`; for more details on the classes see section 2.2).
- The input, which is the return value of `reportBadRegionsSummary` has to be provided (data frame object).
- The folder where the output shall be saved may be defined as well. If it is defined, a png file is saved: "CoverageQuality_Summary.png". If only an empty string is defined, the plot is printed on the screen.

```
> plotSummary(threshold1, threshold2, percentage1, percentage2, badCoverageSummary, "")
```

```
NULL
```

4.2 Detailed variant

If the user likes to visualize the output of `reportBadRegionsDetailed`, it is recommended to use the function `plotDetailed`. A line graph is returned, visualizing the median coverage over all samples at every position of interest. Furthermore, information on whether the base were originally targeted or not, on the coverage quality and the corresponding genes (name and geneID) that are located at the positions is included in the plot.

On the y axis the median coverage over all samples is coded. Every position is considered individually.

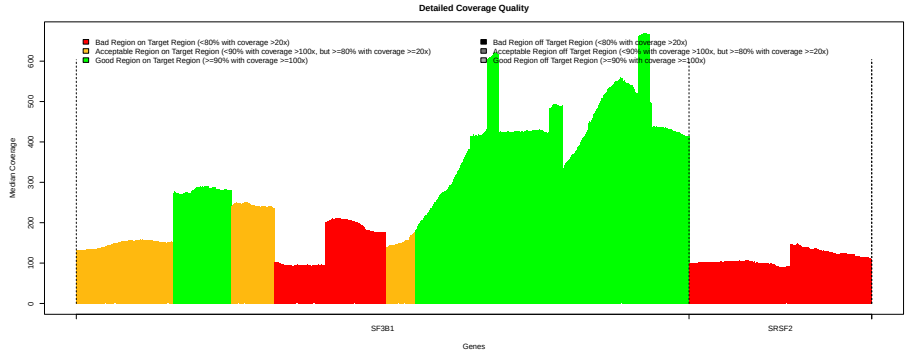
On the x axis the names of the detected genes are printed. Wherever a new region covering a new gene is registered, a dashed line is drawn.

Yet, additionally to the mere median coverage, the corresponding coverage quality at each position is also included in the plot. The different categories of coverage quality are color coded in the following way: red - bad region on target; yellow - acceptable region on target; green - good region on target; black - bad region off target; dark gray - acceptable region off target; light gray - good region off target. Thereby, on- and off target regions may easily be separated as well.

For the correct functioning of `reportBadRegionsSummary` various input objects are necessary:

- Values defining the classes of coverage (`threshold1`, `percentage1`, `threshold2`, `percentage2`; for more details on the classes see section 2.2).
- The input, which is the return value of `reportBadRegionsDetailed` has to be provided (list object).
- The folder where the output shall be saved may be defined as well. If it is defined, a png file is saved: "CoverageQuality_Details.png". If only an empty string is defined, the plot is printed on the screen.

```
> plotDetailed(threshold1, threshold2, percentage1, percentage2, coverage_indicators_temp, "")
```



4.3 Summary variant (genes)

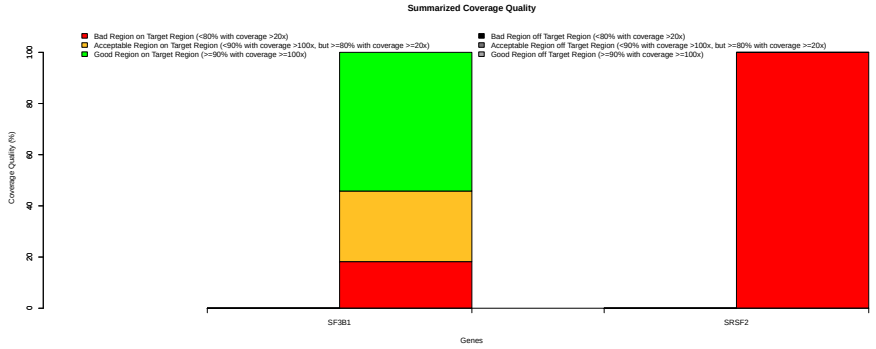
If the user likes to visualize the output of `reportBadRegionsGenes`, it is recommended to use the function `plotSummaryGenes`. The function returns a barplot, visualizing the percentage of each gene that falls into each category of coverage quality. The plot thereby serves to quickly distinguish well from bad covered genes.

For every gene either one or two stacked bars are plotted. If a gene is covered, but it was not originally targeted, a bar is plotted containing the following color code: black - bad region off target; dark gray - acceptable region off target; light gray - good region off target. If a gene was originally targeted, a bar is plotted containing the following color code: red - bad region on target; yellow - acceptable region on target; green - good region on target.

For the correct functioning of `plotSummaryGenes`, various input objects are necessary:

- Values defining the classes of coverage (`threshold1`, `percentage1`, `threshold2`, `percentage2`; for more details on the classes see section 2.2).
- The input, which is the return value of `reportBadRegionsGenes` has to be provided (data frame object).
- The folder where the output shall be saved may be defined as well. If it is defined, a png file is saved: "CoverageQuality_Summary.png". If only an empty string is defined, the plot is printed on the screen.

```
> plotSummaryGenes(threshold1, threshold2, percentage1, percentage2, badCoverageOverview, "")
```



5 Additional functions

5.1 Determine quantiles

Regarding the coverage analysis of many samples in parallel, it may be useful not just to divide the coverage into categories, but to determine certain quantiles over all samples concerning a predefined region of interest. In this case the function `determineQuantiles` may be used.

Considering every chromosome and every position of interest individually, a set of user-defined quantiles is calculated over all samples.

For the correct functioning of `determineQuantiles`, three input objects are necessary:

- The input, which is the return value of `determineCoverage` (with `TRonly=FALSE` or `TRonly=TRUE`) or `determineRegionsOfInterest` (important: no information on the coverage quality) has to be provided (list object).
- The quantiles that should be calculated have to be provided by a vector (e.g. c(0.25,0.5,0.75)).
- The folder where the output shall be saved may be defined. It is saved as: "Quantiles_chr<chromosomename>.txt". If no output shall be written out, an empty string may be passed.

Exemplary return value (chromosome 2):

```
> quantiles <- c(0.5)
> coverage_summary2 <- determineQuantiles(coverage_summary, quantiles, "")
> coverage_summary2[[2]]

GRanges object with 1611 ranges and 2 metadata columns:
         seqnames              ranges strand |       0.5  OnTarget
            <Rle>           <IRanges>  <Rle> | <numeric> <numeric>
     [1]        2         1-198266311      * |         0         0
     [2]        2           198266312      * |        88         0
     [3]        2           198266313      * |        88         0
     [4]        2           198266314      * |        88         0
     [5]        2           198266315      * |        88         0
     ...      ...                 ...    ... .       ...       ...
  [1607]        2           198267917      * |        60         0
  [1608]        2           198267918      * |        60         0
  [1609]        2           198267919      * |        60         0
  [1610]        2           198267920      * |        60         0
  [1611]        2 198267921-243199373      * |         0         0
  -------
  seqinfo: 25 sequences from an unspecified genome; no seqlengths
```

6 Runtime

The runtime of `BadRegionFinder` is highly dependent on the number of reads of the samples to be analyzed and the way they are scattered over the genome. Furthermore, the runtime is of course

also dependent on the number of samples, the length of the genome and the number of bases that are in a *region of interest*.

The step which is most time consuming is certainly the application of the function `determineCoverage`. In the above described examples two samples were analyzed. Alltogether, 1608 bases were located on target (target region consisting of three distinct regions). However, what is much more important, 2174 bases are covered. In the case of the function `determineCoverage` with `TRonly=FALSE`, all covered bases are considered - on and off target. An analysis of the two samples scanning the whole human genome takes 2.207 seconds. If only the two chromosomes which are supposed to be covered (according to the defined target regions) are defined to be considered by the function (`TRonly=TRUE`), the analysis takes 1.93 seconds.

In case the analysis is supposed to consider the coverage in the target region only, usage of the function `determineCoverage` with `TRonly=TRUE` is recommended. The difference in runtime is small, when only few reads are located off target, but it is considerably smaller when the bam files contain many scattered reads.

The runtime of all the other functions is usually considerably smaller. Only in case of the whole genome being analyzed, a high number of scattered reads across the genome and many samples to be analyzed, running these functions may take loner than a minute.

References

[1] S Durinck and W Huber. Interface to biomart databases. http://bioconductor.org/packages/release/bioc/html/biomaRt.html.

[2] A Mammana and J Helmuth. Extract read count signals from bam files. http://www.bioconductor.org/packages/release/bioc/html/bamsignals.html.

[3] AR Quinlan and IM Hall. Bedtools: a flexible suite of utilities for comparing genomic features. *Bioinformatics*, 26:841–842, 2010.

[4] MG Ross, C Russ, M Costello, A Hollinger, NJ Lennon, R Hegarty, C Nusbaum, and DB Jaffe. Characterizing and measuring bias in sequence data. *Genome Biology*, 14(R51), 2013.