TEQC: Target Enrichment Quality Control

Manuela Hummel Sarah Bonnin Ernesto Lowy
Guglielmo Roma

April 14, 2011

Contents

1 Introduction

With whole genome sequencing it is still rather expensive to achieve sufficient read coverage for example for the detection of genomic variants. Further, in some cases one might be interested only in some fraction rather than the whole genome, for example linkage regions or the complete exome. Target capture (target enrichment, targeted sequencing) experiments are a suitable strategy in these situations. The genomic regions of interest are selected and enriched previous to next-generation sequencing. A frequently used application for the enrichment of the target sequences is based on hybridization with pre-designed probes, either on microarrays or in solution. The hybridized molecules are then captured (eluted from the microarrays or pulled-down from the solution, respectively), amplified and sequenced.

Besides quality control of the sequencing data, it is also crucial to assess whether the capture had been successful, i.e. if most of the sequenced reads actually fall on the target, if the targeted bases reach sufficient coverage, and so on. This package provides functionalities to address this issue. Quality measures comprise specificity and sensitivity of the capture, enrichment, per-target read

coverage, coverage uniformity and reproducibility, and read duplicate analysis. The coverage can further be examined for its relation to target region length and GC content of the hybridization probes. The analyses can be based on either single reads or read pairs in case of paired-end sequence data. Results are given as values (e.g. enrichment), tables (e.g. per-target coverage), and several diagnostic plots. The package makes use of data structures and methods from the *IRanges* package, which makes dealing with large sequence data feasible.

TEQC does *not* include general sequencing data quality control (e.g. Phred quality plots), neither tools for sequence alignment. It also does not provide functionalities for follow-up analysis like SNP detection.

2 Load Reads and Targets Data

The (minimum) input needed for the quality control analysis are two files

- A bed file containing the genomic positions (chromosome, start, end) of the targeted regions, one genomic range per line. The targets might be custom designed or commercial solutions, e.g. for the capture of the whole human exome. The file does not have to be sorted with respect to genomic position.
- A bed file containing the genomic positions (chromosome, start, end) of sequenced reads aligned to a reference genome, one genomic range per line. In case of paired-end data an additional column with the read pair ID is suggested. This file format is very general and hence the QC analysis is not limited to any sequencing platform or alignment tool. The file does not have to be sorted with respect to genomic position.

The package includes a small example data set. First we load the target positions. The input *targetsfile* does not need to have a fixed format. Just three columns containing chromosome (as string, e.g. `chr1`), start and end position of each target are required. The options *chrcol*, *startcol* and *endcol* specify in which columns of the file the respective information is found. The output of `get.targets` is of class *RangedData* from the *IRanges* package. Note that overlapping or adjacent targets are merged, such that the returned target regions are not-overlapping. Therefore, also a bed file containing information about the hybridization probes, which might be highly overlapping due to tiling, can be used as *targetsfile*.

```
> library(TEQC)
> exptPath <- system.file("extdata", package = "TEQC")
> targets <- get.targets(targetsfile = paste(exptPath, "ExampleSet_Targets.bed",
+     sep = "/"), chrcol = 1, startcol = 2, endcol = 3, skip = 0)

[1] "read 50 (non-overlapping) target regions"

> targets

RangedData with 50 rows and 0 value columns across 21 spaces
       space                 ranges   |
    <factor>              <IRanges>   |
```

```
1       chr1 [ 11158025,  11158264]   |
2       chr1 [ 25870174,  25870293]   |
3       chr1 [ 65656333,  65656572]   |
4       chr1 [ 68611504,  68611743]   |
5       chr1 [ 70225862,  70226101]   |
6       chr1 [112269528, 112270487]   |
7       chr1 [160970804, 160970923]   |
8       chr1 [182635998, 182636117]   |
9       chr1 [186270664, 186270903]   |
...      ...                     ... ...
42      chr6 [152787105, 152787224]   |
43      chr6 [170639528, 170639647]   |
44      chr7 [144345876, 144345995]   |
45      chr8 [  1830798,   1830917]   |
46      chr9 [  6241680,   6241799]   |
47      chr9 [ 77277336,  77277575]   |
48      chr9 [ 79827855,  79827974]   |
49      chrX [ 47086386,  47086505]   |
50      chrX [150891046, 150891285]   |
```

NOTE: We assume that genomic positions in bed files follow the 0-based start / 1-based end coordinate system as defined by UCSC (`http://genome.ucsc.edu/FAQ/FAQformat`). In *TEQC* we need 1-based coordinates, so by default `get.targets` and `get.reads` (see later) shift all start positions forward by 1. If the coordinates in your files are already 1-based, set the parameter *zerobased* to *FALSE* in order to avoid the shifting.

We might ask what fraction of the genome is targeted. In the function `fraction.target` the corresponding genome can be specified by the option *genome*. At the moment only *hg18* and *hg19* are available. The corresponding genome sizes are taken from `http://genomewiki.ucsc.edu/index.php/Genome_size_statistics`. For any other case, you can specify the genome size manually with the option *genomesize*. In our little example the total targeted region and hence the fraction within the genome is very small.

```
> ft <- fraction.target(targets, genome = "hg19")
> ft

[1] 4.158855e-06
```

Next, we load the genomic positions of the aligned reads. Depending on the number of reads, this can be quite time and memory consuming with real data. The function `get.reads` is quite similar to `get.targets`. However, overlapping or identical reads are not merged. Furthermore, a column containing read identifiers can be specified with option *idcol*. This is essential in case of paired-end data, when you want quality statistics to be done (also) on read pairs rather than on single reads. In this case the ID has to be the identifier of the read pair (i.e. the same unique ID for both reads of the pair). Our example data was derived by paired-end sequencing, so we keep the pair IDs in the resulting *RangedData* object. The genomic positions in our bedfile containing the reads are 1-based, so we avoid coordinate shifting by setting *zerobased* to *FALSE*.

```
> reads <- get.reads(paste(exptPath, "ExampleSet_Reads.bed", sep = "/"),
+     chrcol = 1, startcol = 2, endcol = 3, idcol = 4, zerobased = F,
+     skip = 0)

[1] "read 19546 sequenced reads"

> reads

RangedData with 19546 rows and 1 value column across 24 spaces
         space                 ranges   |                ID
      <factor>              <IRanges>   |       <character>
1         chr1     [  13328,   13381]   |    1_16_7090_2464
2         chr1     [  13467,   13520]   |    1_16_7090_2464
3         chr1     [1420325, 1420378]   |    1_99_1631_6326
4         chr1     [1420402, 1420455]   |    1_99_1631_6326
5         chr1     [2321365, 2321418]   |   1_5_14614_17275
6         chr1     [2321479, 2321532]   |   1_5_14614_17275
7         chr1     [2452643, 2452696]   |   1_15_18642_6232
8         chr1     [2452775, 2452828]   |   1_15_18642_6232
9         chr1     [2535217, 2535270]   |    1_20_2015_5490
...        ...                    ... ...               ...
19538     chrX [154014582, 154014635]   |    1_6_7123_16490
19539     chrX [154261641, 154261694]   |    1_88_4216_1073
19540     chrX [154261777, 154261830]   |    1_88_4216_1073
19541     chrY [  3551497,   3551550]   |    1_87_3763_6007
19542     chrY [  3551666,   3551719]   |    1_87_3763_6007
19543     chrY [ 10028334,  10028387]   |   1_63_1439_10606
19544     chrY [ 10028434,  10028487]   |   1_63_1439_10606
19545     chrY [ 28425424,  28425477]   | 1_105_17963_15521
19546     chrY [ 28425590,  28425643]   | 1_105_17963_15521
```

Paired-end data

When reads are paired, in order to perform statistics on pairs rather than on single reads, the read pairs have to be matched together using function `reads2pairs`. To run the function can be quite time consuming, depending on the number of reads. The output is a *RangedData* object whose ranges start at the first base of the first read within a read pair and end at the last base of the respective second read. This is equivalent to the positions of the DNA molecule that was actually sequenced from both ends. The *reads* table might also contain single reads (i.e. whose corresponding partners did not align to the reference). In this case a list of two *RangedData* tables will be returned, the first one containing the original positions of the single reads without partners (table `singleReads`), and the other one containing the merged pairs positions (table `readpairs`). The provided reads data might also contain cases where the two reads of a pair align to different chromosomes. Since for such read pairs a 'merging' does not make sense, they will be returned within the `singleReads` table. Further, for some pairs the respective reads might align very far apart within the same chromosome. If you wish to remove such reads, you can specify a value for option *max.distance*. Reads with a larger distance (from start position of first read to end position of second read) will be added to table `singleReads`.

```
> readpairs <- reads2pairs(reads)
> readpairs

RangedData with 9773 rows and 1 value column across 24 spaces
        space                 ranges   |                ID
     <factor>              <IRanges>   |       <character>
1        chr1     [  13328,   13520]   |    1_16_7090_2464
2        chr1     [1420325, 1420455]   |    1_99_1631_6326
3        chr1     [2321365, 2321532]   |   1_5_14614_17275
4        chr1     [2452643, 2452828]   |   1_15_18642_6232
5        chr1     [2535217, 2535407]   |    1_20_2015_5490
6        chr1     [2814194, 2814357]   |  1_36_17959_12842
7        chr1     [6488248, 6488458]   |  1_58_10090_11513
8        chr1     [6741034, 6741168]   |    1_66_3980_7233
9        chr1     [7332529, 7332654]   |    1_85_5028_4821
...       ...                   ... ...                ...
9765     chrX [153700924, 153701112]   |   1_26_10299_1105
9766     chrX [153814666, 153814897]   |  1_74_16848_10502
9767     chrX [153851675, 153851897]   |    1_14_5061_8480
9768     chrX [153997431, 153997581]   |  1_100_7570_16280
9769     chrX [154014446, 154014635]   |    1_6_7123_16490
9770     chrX [154261641, 154261830]   |    1_88_4216_1073
9771     chrY [  3551497,   3551719]   |    1_87_3763_6007
9772     chrY [ 10028334,  10028487]   |   1_63_1439_10606
9773     chrY [ 28425424,  28425643]   | 1_105_17963_15521
```

Again, only for the case of paired-end data, we can visualize the read pair insert sizes, i.e. the distances from the start of read 1 to the end of read 2, respectively. For the function `insert.size.hist` we need the output of the previous call to `reads2pairs`. In our example, the alignment was done in a way that reads were only kept if the two reads aligned at a maximum distance of 250 bases. Therefore, the resulting histogram is truncated at 250, see figure 1. Note that also average and median insert sizes displayed in the graph are based on the truncated data, and hence are not statistics for the true read pair distribution. The insert sizes for all read pairs can also be returned using option *returnInserts = TRUE*.

```
> insert.size.hist(readpairs, breaks=10)
```

3 Specificity and Enrichment

One important component of quality control in target capture experiments is to check whether most of the sequenced reads actually fall on target regions. A barplot showing the numbers of reads aligning to each chromosome can give a first impression on that. When providing the function `chrom.barplot` only with the reads table, the resulting barplot will show absolute counts. There is also the option to give both the reads and the targets table, which will show fractions of reads and targets, respectively, falling on each chromosome. For the reads, this is the fraction within the total *number* of reads (since reads are usually expected to have all the same length). In contrast, for the targets,

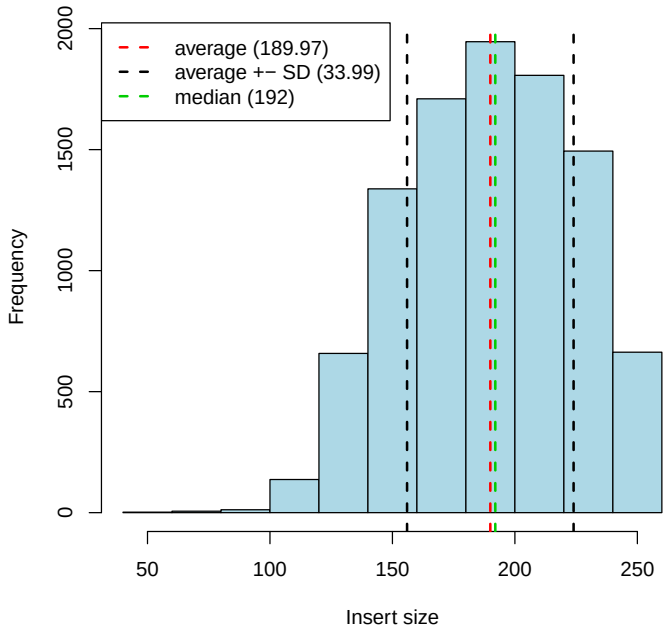

Figure 1: Histogram of read pair insert sizes.

the fraction of targeted bases on each chromosome is calculated. Since targets might strongly vary in length it is reasonable to account for the actual target *sizes* instead of considering merely numbers of targets per chromosome. In this way you can compare directly if the amount of reads corresponds more or less to the amount of target on a certain chromosome (see figure 2).

```
> chrom.barplot(reads, targets)
```

A measure for the capture specificity is the fraction of aligned reads that overlap with any target region. It can be calculated by function `fraction.reads.target`. The function has an option *mappingReads* that can be set to *TRUE* in order to retrieve a reduced `reads` *RangedData* table containing only those reads overlapping target regions.

```
> fr <- fraction.reads.target(reads, targets)
> fr
```

```
[1] 0.3899007
```

In many of the functions within *TEQC* you can specify an "offset" that will enlarge every target on each side by the specified number of bases. Since usually the captured DNA molecules are longer than what is actually sequenced, it is expected to have many reads that do not overlap, but are close to the target. Considering e.g. the actual targets plus 100 bases on each side, we get a higher on-target fraction:

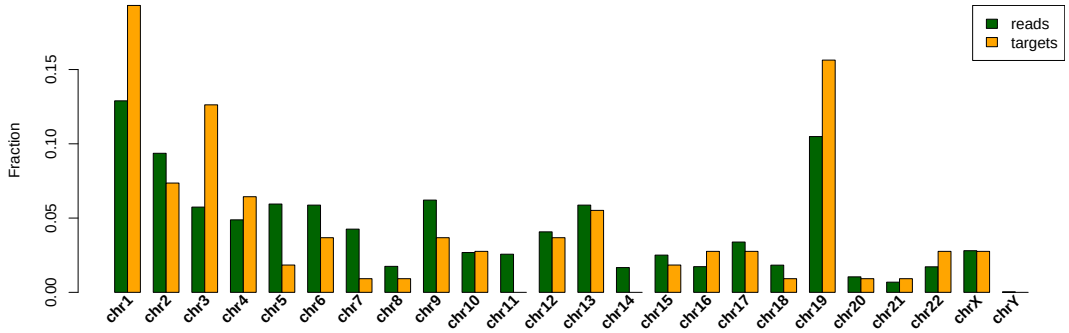

Figure 2: Fractions of reads (green) and target (orange) per chromosome.

```
> fraction.reads.target(reads, targets, Offset = 100)
```

```
[1] 0.4853679
```

With the fraction of on-target reads and the fraction of the targeted region within the reference genome, we can calculate the *enrichment*

$$\text{enrichment} = \frac{\#\text{ reads on target}/\#\text{ aligned reads}}{\text{target size}/\text{genome size}}$$

Since in our artificial example the total target size is unrealistically small, we achieve a huge enrichment.

```
> fr/ft
```

```
[1] 93751.94
```

Instead of considering single reads, we could also calculate the fraction of read pairs that are on-target. A read pair is counted as on-target if at least one of its reads overlaps with a target region or, in case of small targets, if the first read lies "left" and the second read "right" of the target and hence the corresponding sequenced molecule covered the target completely. For the specificity calculation the same function can be used, just the input changes from the table containing all reads to the table created above by `reads2pairs`.

```
> fraction.reads.target(readpairs, targets)
```

```
[1] 0.4934002
```

4 Coverage

Besides high capture specificity, it is of course important to check the read coverage within target regions, since it is crucial for follow-up analyses. The function `coverage.target` calculates read coverage for each base that is sequenced

and/or located in a target region. It returns a list with the average (list element `avgTargetCoverage`), standard deviation (`targetCoverageSD`) and main quantiles (`targetCoverageQuantiles`) of the coverage over all targeted bases. When option *perBase* is set to *TRUE*, the returned list additionally has the two elements `coverageAll` and `coverageTarget`. The former one is a *SimpleRleList* containing coverages for all bases present in the `reads` table, the latter one contains coverages for all targeted bases. For some *TEQC* functions we need either one or the other. When option *perTarget* is set to *TRUE*, the returned list has the additional element `targetCoverages`. This is the input `targets` *RangedData* table, now including as 'values' columns the average coverage (column `avgCoverage`) and standard deviation (column `coverageSD`) per target region. Coverage calculations might take a while, depending on the numbers of reads and targets.

```
> Coverage <- coverage.target(reads, targets, perTarget = T, perBase = T)
> Coverage

$avgTargetCoverage
[1] 27.47927

$targetCoverageSD
[1] 23.68119

$targetCoverageQuantiles
  0%  25%  50%  75% 100%
   0    8   21   42  117

$targetCoverages
RangedData with 50 rows and 2 value columns across 21 spaces
       space               ranges   | avgCoverage coverageSD
    <factor>            <IRanges>   |   <numeric>  <numeric>
1       chr1 [ 11158025,  11158264]   |    44.12917  10.088934
2       chr1 [ 25870174,  25870293]   |     0.00000   0.000000
3       chr1 [ 65656333,  65656572]   |    31.75833   6.931217
4       chr1 [ 68611504,  68611743]   |    28.79583  13.489341
5       chr1 [ 70225862,  70226101]   |    24.16667   8.049360
6       chr1 [112269528, 112270487]   |    10.14479   3.755132
7       chr1 [160970804, 160970923]   |    16.25833   3.074689
8       chr1 [182635998, 182636117]   |    33.85833   5.987797
9       chr1 [186270664, 186270903]   |    32.48333   7.509529
...      ...                   ... ...         ...         ...
42      chr6 [152787105, 152787224]   |  45.0750000 13.6708993
43      chr6 [170639528, 170639647]   |  31.1916667  9.0834560
44      chr7 [144345876, 144345995]   |  78.3250000  7.9061326
45      chr8 [  1830798,   1830917]   |   0.2166667  0.4137009
46      chr9 [  6241680,   6241799]   |  42.5083333  5.7203688
47      chr9 [ 77277336,  77277575]   |  72.5125000 18.9239794
48      chr9 [ 79827855,  79827974]   |  26.1250000  3.8731190
49      chrX [ 47086386,  47086505]   |  14.5916667  2.8269288
50      chrX [150891046, 150891285]   |  10.0083333  2.7244310
```

```
$coverageAll
SimpleRleList of length 24
$chr1
'integer' Rle of length 248737162 with 3785 runs
  Lengths:   13327      54      85      54 ...  185876      54     130      54
  Values :       0       1       0       1 ...       0       1       0       1

$chr10
'integer' Rle of length 135113702 with 1032 runs
  Lengths:  294227      54     116      54 ...      37      17      14      54
  Values :       0       1       0       1 ...       2       1       0       1

$chr11
'integer' Rle of length 134010621 with 1004 runs
  Lengths:  233012      54      14      54 ...  335161      54      85      54
  Values :       0       1       0       1 ...       0       1       0       1

$chr12
'integer' Rle of length 133740173 with 1352 runs
  Lengths:  274541      54      37      54 ...   56585      54      76      54
  Values :       0       1       0       1 ...       0       1       0       1

$chr13
'integer' Rle of length 115007810 with 1218 runs
  Lengths: 19409537       54       81       54 ...       54       92       54
  Values :        0        1        0        1 ...        1        0        1

...
<19 more elements>

$coverageTarget
SimpleRleList of length 21
$chr1
'integer' Rle of length 2520 with 1079 runs
  Lengths:  1  1  1  1  2  2  1  2  1  1  1 ...  1  1  2  1  1  2  1  1  1  1
  Values : 23 22 21 23 24 23 24 21 22 21 22 ... 39 40 38 37 36 35 34 33 34 33

$chr10
'integer' Rle of length 360 with 62 runs
  Lengths:  8 12  1  1  5  1  2  5  6  3  1 ...  9  2  7 21 11  2 13  7 11 13
  Values :  6  7  8  9  8  9  8  7  6  5  6 ...  4  3  4  3  2  3  2  1  2  3

$chr12
'integer' Rle of length 480 with 201 runs
  Lengths:  1  2  2  2  2  1  1  4  1  2  4 ...  1  4  3  5  5  3  1  3  2  9
  Values : 18 20 22 20 21 20 22 21 23 24 25 ...  3  4  5  6  7  8  7  8 10  9

$chr13
'integer' Rle of length 720 with 491 runs
```

```
  Lengths:  1  1  1  2  1  5  2  1  1  1  3 ...  1  1  3  2  2  3  8  1  1  1
  Values : 32 31 32 33 37 38 37 39 43 41 43 ... 25 23 21 22 25 24 23 22 24 25

$chr15
'integer' Rle of length 240 with 102 runs
  Lengths:  9  1  1  9  1  6  2  5  3  3  5 ...  2  2  1  1  2  2  3  1  5  4
  Values : 14 13 12 11 10  9 10  9 10 11 12 ... 19 18 17 15 16 17 16 15 16 17

...
<16 more elements>
```

```
> targets2 <- Coverage$targetCoverages
```

A different measure for the per-target coverage would be to count the numbers of reads overlapping with each target. This can be calculated by the function `readsPerTarget`. It returns again the input `targets` table while adding a 'values' column that gives the respective numbers of reads per target. If we provide the table `targets2` from above, we will get both the average per-target coverages and the numbers of target-overlapping reads in the same table. In order to speed things up, instead of all `reads` we could also provide the `mappingReads` output of function `fraction.reads.target`, since for counting the reads per target it is enough to look at the reads from which we already know they are on-target. Just make sure that the *Offset* option was set the same in both `fraction.reads.target` and `readsPerTarget`.

```
> targets2 <- readsPerTarget(reads, targets2)
> targets2
```

```
RangedData with 50 rows and 3 value columns across 21 spaces
       space                 ranges   | avgCoverage coverageSD    nReads
    <factor>              <IRanges>   |   <numeric>  <numeric> <numeric>
1       chr1 [ 11158025,  11158264]   |    44.12917  10.088934       231
2       chr1 [ 25870174,  25870293]   |     0.00000   0.000000         0
3       chr1 [ 65656333,  65656572]   |    31.75833   6.931217       164
4       chr1 [ 68611504,  68611743]   |    28.79583  13.489341       152
5       chr1 [ 70225862,  70226101]   |    24.16667   8.049360       126
6       chr1 [112269528, 112270487]   |    10.14479   3.755132       185
7       chr1 [160970804, 160970923]   |    16.25833   3.074689        50
8       chr1 [182635998, 182636117]   |    33.85833   5.987797        90
9       chr1 [186270664, 186270903]   |    32.48333   7.509529       166
...      ...                   ... ...         ...         ...       ...
42      chr6 [152787105, 152787224]   |  45.0750000 13.6708993       129
43      chr6 [170639528, 170639647]   |  31.1916667  9.0834560        84
44      chr7 [144345876, 144345995]   |  78.3250000  7.9061326       241
45      chr8 [  1830798,   1830917]   |   0.2166667  0.4137009         1
46      chr9 [  6241680,   6241799]   |  42.5083333  5.7203688       124
47      chr9 [ 77277336,  77277575]   |  72.5125000 18.9239794       369
48      chr9 [ 79827855,  79827974]   |  26.1250000  3.8731190        84
49      chrX [ 47086386,  47086505]   |  14.5916667  2.8269288        46
50      chrX [150891046, 150891285]   |  10.0083333  2.7244310        51
```

The resulting *RangedData* table can be converted to a data frame, and as such easily be written to a file, e.g. by

```
> write.table(as.data.frame(targets2), file="target_coverage.txt",
sep="\t", row.names=F, quote=F)
```

Talking about coverage, it is interesting to ask which fraction of target bases reach a coverage of at least k (some value relevant for further analyses, e.g. SNP calling) or which fraction of target bases is covered at all by any read (sensitivity of the capture). The function `covered.k` calculates such fractions based on the `coverageTarget` output of `coverage.target`. Option k specifies the values for which to calculate the fraction of bases achieving the respective coverage.

```
> covered.k(Coverage$coverageTarget, k = c(1, 5, 10))

        1         5        10
0.9520963 0.8444087 0.7033801
```

With `coverage.hist` we can visualize the coverage distribution, see figure 3. A line is added to the histogram that shows the cumulative fraction of target bases with a coverage of at least the corresponding x-axis value. The line represents the results of `covered.k` for all possible values of k. Additionally, you can highlight with dashed lines the base fraction achieving a coverage of at least a certain value by defining the option *covthreshold*.

```
> coverage.hist(Coverage$coverageTarget, covthreshold=8)
```

A similar graph is the coverage uniformity plot, see figure 4. It corresponds more or less to the cumulative line in the coverage histogram. However, `coverage.uniformity` calculates normalized coverages, i.e. the per-base coverages divided by the average coverage over all target bases. Normalized coverages are not dependent on the absolute quantity of sequenced reads and are hence better comparable between different samples or even different experiments. By default, the x-axis in the figure is truncated at 1, which corresponds to the average normalized coverage. The steeper the curve is falling, the less uniform is the coverage.

```
> coverage.uniformity(Coverage)
```

There are more graphical functions concerning read coverage. For example you might be interested in whether large targets are covered by more reads, as expected, since for larger target regions there should also be more hybridization capture probes ("baits"). Or you might ask whether quite small targets have worse coverage, because the bait tiling might not be as good as for larger targets. Those questions can be addressed by the function `coverage.targetlength.plot`. As input a *RangedData* targets table has to be given that contains the relevant information in the 'values' column(s). The graphs are useful for targets of very different lengths. In our example, where all the targets are rather small, the figures are not very informative, see figure 5.

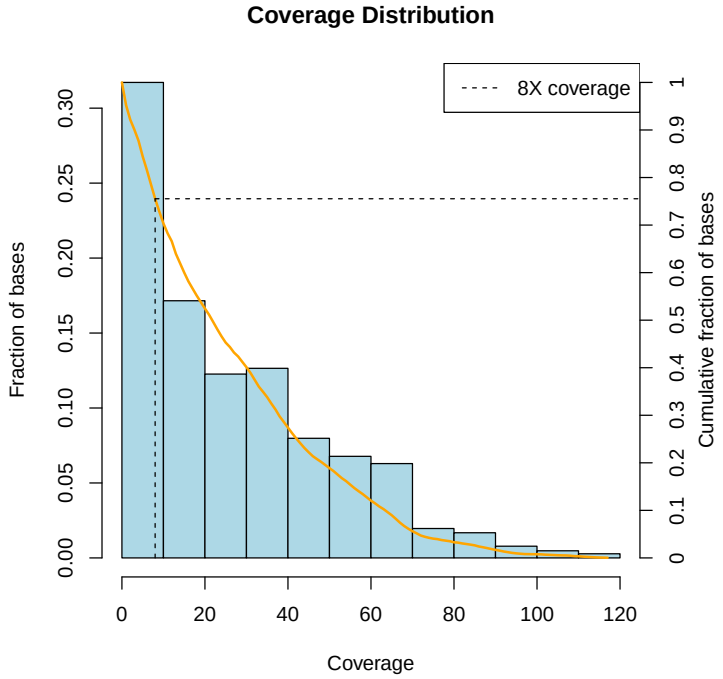

Figure 3: On-target coverage histogram. The orange line shows the cumulative fraction of target bases (right y-axis) with a read coverage of at least x. The dashed lines highlight the fraction of target bases covered by at least 8 reads.

```
> par(mfrow=c(1,2))
> coverage.targetlength.plot(targets2, plotcolumn="nReads", pch=16,
cex=1.5)
> coverage.targetlength.plot(targets2, plotcolumn="avgCoverage", pch=16,
cex=1.5)
```

Another thing to check is the dependency between coverage and GC content of the hybridization capture probes ("baits"). For calculating the GC contents the bait sequences are needed. A file has to be created beforehand that contains the positions as well as the sequences of all baits. The file is loaded by function `get.baits` which is similar to `get.targets`, with the difference that overlapping or adjacent baits are not merged, and that a column *seqcol* has to be specified that holds the bait sequences. Like `get.targets` and `get.reads`, also `get.baits` by default converts the bait start positions from 0-based to 1-based coordinates. If the positions given in your `baitsfile` are already 1-based, set option *zerobased* to *FALSE*.

```
> baitsfile <- paste(exptPath, "ExampleSet_Baits.txt", sep = "/")
> baits <- get.baits(baitsfile, chrcol = 3, startcol = 4, endcol = 5,
+     seqcol = 2)

[1] "read 108 hybridization probes"
```

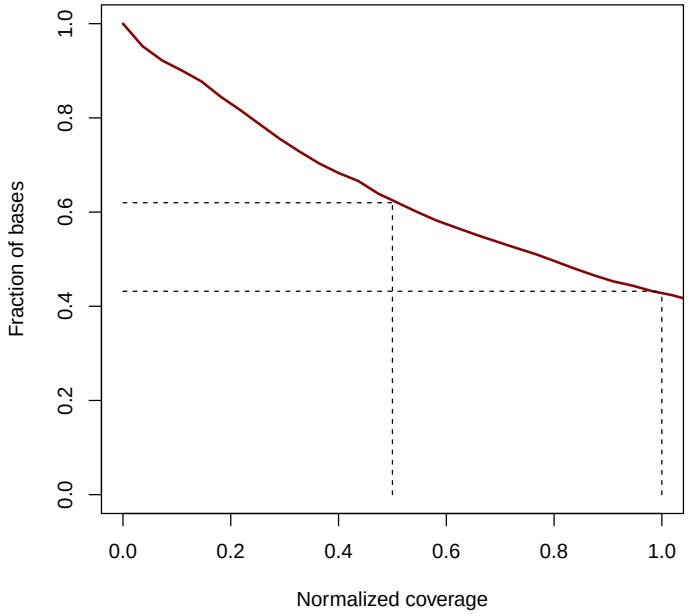

Figure 4: Fraction of targeted bases (y-axis) achieving a normalized coverage of at least x. Dashed lines indicate the fractions of bases achieving at least the average (= 1) or at least half the average coverage (= 0.5).

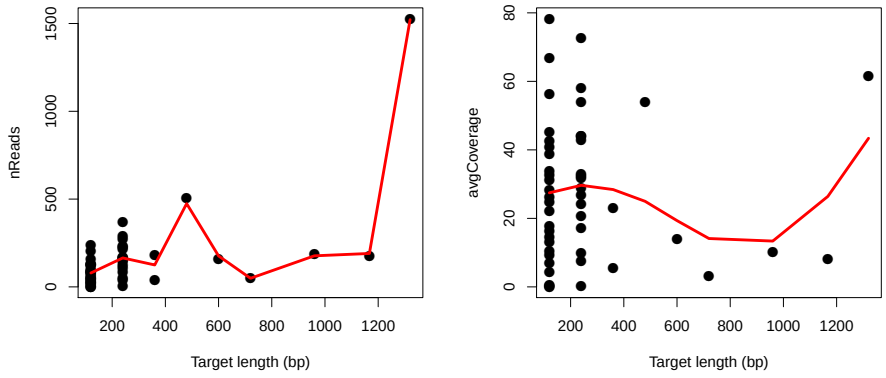

Figure 5: Scatter plots and smoothing splines of number of reads per target (left panel) or average coverage per target (right panel) versus respective target length.

With the `baits` *RangedData* table and the `coverageAll` output of `coverage.target` the (normalized) coverage versus GC content plot can be created.

This can take quite some time, since GC content and average (normalized) coverage have to be calculated for every bait. The bait coverages can be returned by setting option *returnBaitValues* = *TRUE*. You would expect the added smoothing spline to have an inverse U-shape, with a peak in coverage for baits with GC content around 40-50%. In our small example there are not enough baits with low GC content to encounter the expected shape, see figure 6.

```
> coverage.GC(Coverage$coverageAll, baits, pch=16, cex=1.5)
```

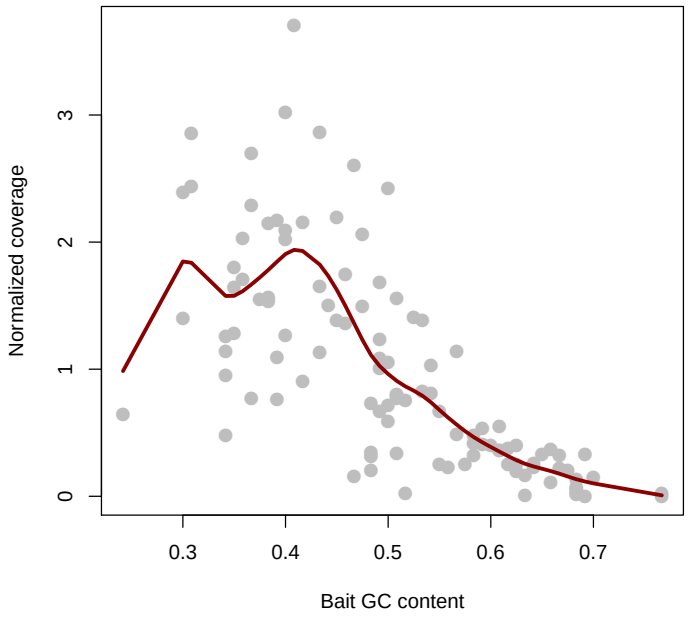

Figure 6: Scatter plot and smoothing spline of normalized average coverage per hybridization probe versus GC content of the respective bait.

There is also a function `coverage.plot` to visualize per-base coverages along chromosomal positions. The input has to be the `coverageAll` output of function `coverage.target`, since also coverages of off-target bases are needed. The positions of target regions, potentially extended on both sides, can be highlighted as well by specifying options *targets* and *Offset* (see figure 7).

```
> coverage.plot(Coverage$coverageAll, targets, Offset=100, chr="chr1",
Start=11157524, End=11158764)
```

Of course, coverages can also be visualized by genome browsers. We provide the function `make.wigfiles` to create wiggle files that can then be uploaded e.g. to the UCSC genome browser. You can make wiggle files for all chromosomes on which there are reads or just for some selected chromosome(s) by specifying

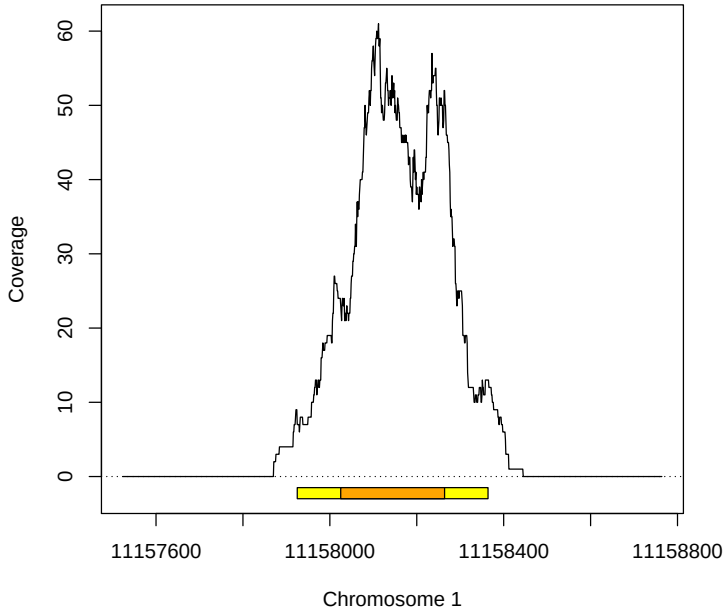

Figure 7: Per-base coverages along chromosomal positions. Target regions (plus addition of 100 bases on both sides) are highlighted in orange (yellow).

option *chroms*. With option *filename* the name and location where to save the files can be manipulated. Base positions are given as 1-based (i.e. the first base on a chromsome has position 1), as defined for wiggle files by UCSC (`http://genome.ucsc.edu/goldenPath/help/wiggle.html`).

```
> make.wigfiles(Coverage$coverageAll)
```

5 Read Duplicates

A crucial issue in target capture experiments is read duplication. Usually, read duplicates, i.e. reads that have exact same start and end positions, are removed before follow-up analysis of sequencing data, since they are supposed to be PCR artifacts. However, here we expect a probably substantial amount of "real" read duplication due to the enrichment process. "Real" read duplicates would be derived from actually separate input DNA molecules that by chance were fragmented at the same position. The `duplicates.barplot` shows which fraction of reads / read pairs is present in the data in what number of copies. Read multiplicity proportions are calculated and shown separately for on- and off-target reads / read pairs. Therefore, the plot gives an impression about the amount of "real" duplication (expected mostly in the target regions) versus artifactual duplication (expected both on- and off-target). With option *returnDups = TRUE* the absolute numbers (given on top of the bars, in millions) and percentages

(bar heights) can be returned.

```
> duplicates.barplot(reads, targets)
```

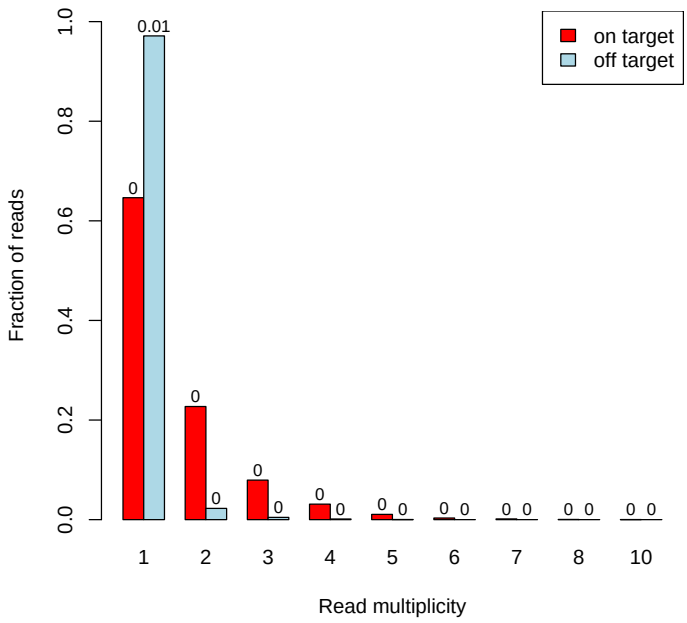

Figure 8: Duplicates barplot treating data as single-end. It shows fractions of on- and off-target reads, respectively, that are unique, that are there in two copies, three copies, etc. (x-axis). The numbers on top of the bars are absolute counts in millions.

In figure 8 we see, firstly, that for some reads there are quite a lot of identical copies (x-axis up to 10). Secondly, the percentage of reads with multiple copies is much higher within on-target reads than within off-target reads (red bars are much higher than blue ones for $x > 1$). This suggest, as mentioned before, that there might be substantial amount of "real" read duplication.

In the case of paired-end data, the position information of both reads of a pair can be used. Reads only have to be considered duplicated if the positions of both reads are found again in another pair. We can use `duplicates.barplot` with the table `readpairs` we created before to make the graph for read pairs instead of single reads. For read pairs the extent of duplication is by far not as high as for single reads, see figure 9.

```
> duplicates.barplot(readpairs, targets, ylab="Fraction of read pairs")
```

Unfortunately, it is not possible to distinguish the artfactual duplicates from the naturally occurring ones. For that reason it might still be recommendable to remove duplication before further analysis by keeping each duplicated read

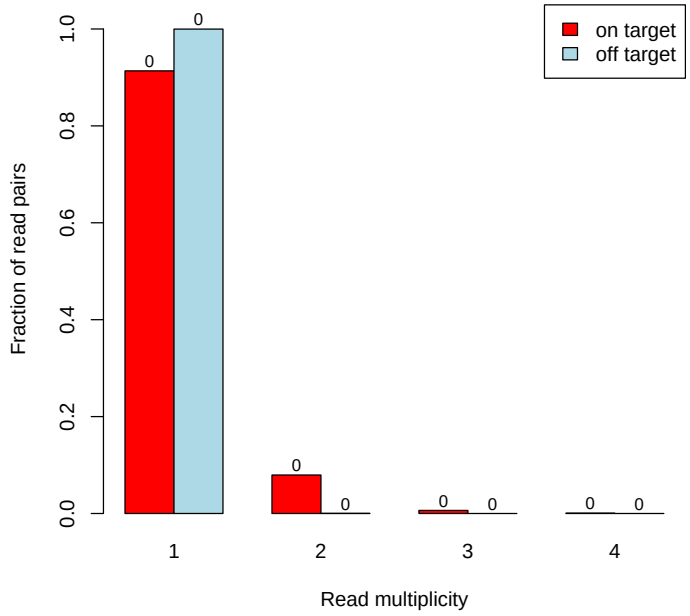

Figure 9: Duplicates barplot for read pairs.

position only once. (Since here we just deal with genomic positions and not with the actual reads that might differ in sequence and quality, it does not matter *which* copy of a duplicated read to keep.) You might also decide to do all the QC analysis on the "collapsed" data. Either a bed file is provided that includes only unique read positions, or the tables are collapsed within `R`, e.g. by the following code.

```
> dupfun <- function(x) duplicated(x$ranges)
> params <- RDApplyParams(rangedData = reads, applyFun = dupfun)
> dups <- unlist(rdapply(params))
> reads.collapsed <- reads[!dups, , drop = T]
```

In the case of paired-end data, as said before, reads only have to be considered duplicated if the positions of both reads are found again in another pair. Removing duplicates can be done like follows, selecting non-duplicated reads from the `readpairs` table (and in case of duplication just keep one of the respective reads) and extracting those from the original `read` table.

```
> params2 <- RDApplyParams(rangedData = readpairs, applyFun = dupfun)
> dups2 <- unlist(rdapply(params2))
> ID.nondups <- readpairs$ID[!dups2]
> sel <- reads$ID %in% ID.nondups
> reads.collapsed.pairs <- reads[sel, , drop = T]
```

In single-end data it is probable to lose a large number of reads by removing duplicates. When treating our example data like single-end sequences, we are left with only 16417 out of originally 19546 reads. Hence, the actual coverage could decrease dramatically.

```
> coverage.target(reads.collapsed, targets, perBase = F, perTarget = F)

$avgTargetCoverage
[1] 17.43535

$targetCoverageSD
[1] 11.7878

$targetCoverageQuantiles
  0%  25%  50%  75% 100%
   0    7   16   27   46
```

Before removing duplicates we calculated a coverage of 27.5.

When using position information of both reads per pair, much less reads are lost, e.g. in the example we can keep 18714 reads. Therefore, the coverage will not decrease that much.

```
> coverage.target(reads.collapsed.pairs, targets, perBase = F,
+     perTarget = F)

$avgTargetCoverage
[1] 25.09604

$targetCoverageSD
[1] 21.22714

$targetCoverageQuantiles
  0%  25%  50%  75% 100%
   0    7   20   38  110
```

6 Reproducibility

For any new technology, reproducibility is an important issue. Here we base the reproducibility check on per-target-base coverages. Especially for technical replicates we should yield similar results. But the following graphs might also be useful to ensure homogeneity across biological replicates.

To give an example, we create an artificial new sample by removing randomly 10% of the reads from our data.

```
> r <- sample(nrow(reads), 0.1 * nrow(reads))
> reads2 <- reads[-r, , drop = T]
> Coverage2 <- coverage.target(reads2, targets, perBase = T)
```

With function `coverage.density` the coverage densities of several samples can be compared. With option *normalized* you can choose whether to plot original or normalized coverages. When plotting original values in the example, it is

obvious that the second sample does not reach as high coverage for many bases as the first one, see figure 10, right panel. In contrast, normalized coverages are not dependent on the total amount of reads, and we observe a very similar coverage distribution for both samples (left panel), as expected since we just removed some reads randomly.

```
> covlist <- list(Coverage, Coverage2)
> par(mfrow=c(1,2))
> coverage.density(covlist)
> coverage.density(covlist, normalized=F)
```

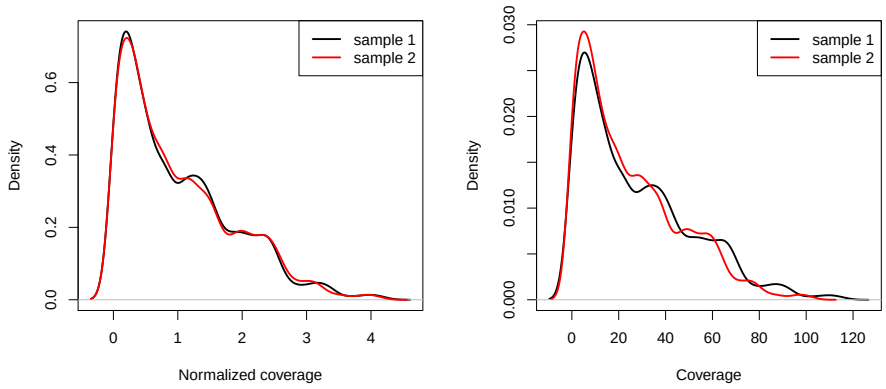

Figure 10: Target coverage densities of two samples, using normalized (left) or original (right) coverages.

Also the coverage uniformity and coverage along chromosome plots shown above (figures 4 and 7) can be produced for several samples within the same graph. The functions can be called repeatedly, while specifying option *add=TRUE* (see figures 11 and 12). As for the densities, the uniformity is almost identical for the two samples since normalized coverage values are used for this plot.

```
> coverage.uniformity(Coverage, addlines=F)
> coverage.uniformity(Coverage2, addlines=F, add=T, col="blue", lty=2)

> coverage.plot(Coverage$coverageAll, targets, Offset=100, chr="chr1",
Start=11157524, End=11158764)
> coverage.plot(Coverage2$coverageAll, add=T, col.line=2, chr="chr1",
Start=11157524, End=11158764)
```

With function `coverage.correlation` we can produce scatterplots between coverage values of pairs of replicate samples. Since a scatterplot for all per-target-base coverages would be huge (e.g. as pdf graph), and moreover the difference between one million or ten million points in the graphic might not be visible at all, just some fraction of randomly selected values is displayed.

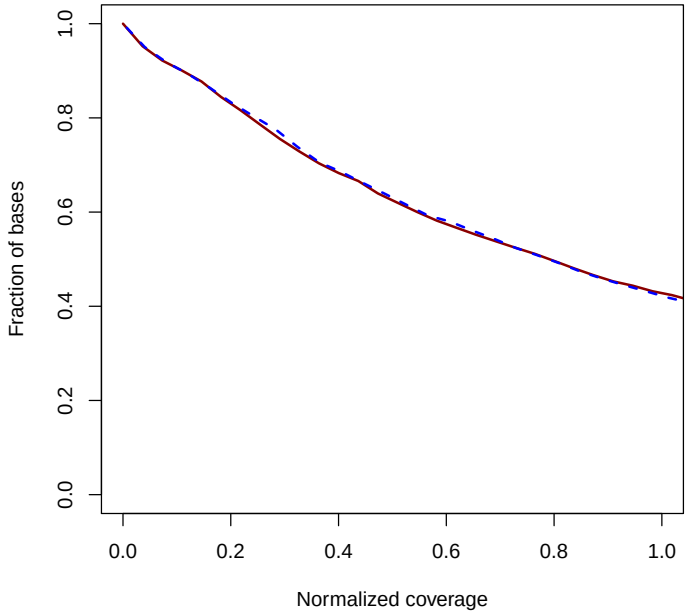

Figure 11: Target coverage uniformity of two samples.

The amount of points to plot can be controlled by option *plotfrac*. By default, 0.1% of all targeted bases are taken into account. In the example we set the fraction to 10%, see figure 13. Scatterplots are shown in the style of a `pairs` plot. In the lower panels the corresponding Pearson correlation coefficients are shown. Correlation calculations are always based on all coverage values, even if *plotfrac* < 1 is chosen. Like in `coverage.density`, by option *normalized* it can be chosen whether to plot normalized or original coverage values.

```
> coverage.correlation(covlist, plotfrac=0.1, cex.pch=4)
```

7 Acknowledgement

The example data used in this manual was taken from an exome sequencing data set of Raquel Rabionet and Xavier Estivill.

8 Session Information

```
> toLatex(sessionInfo())
```

- R version 2.13.0 (2011-04-13), `x86_64-unknown-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_US.UTF-8`, `LC_COLLATE=C`, `LC_MONETARY=C`, `LC_MESSAGES=en_US.UTF-8`,

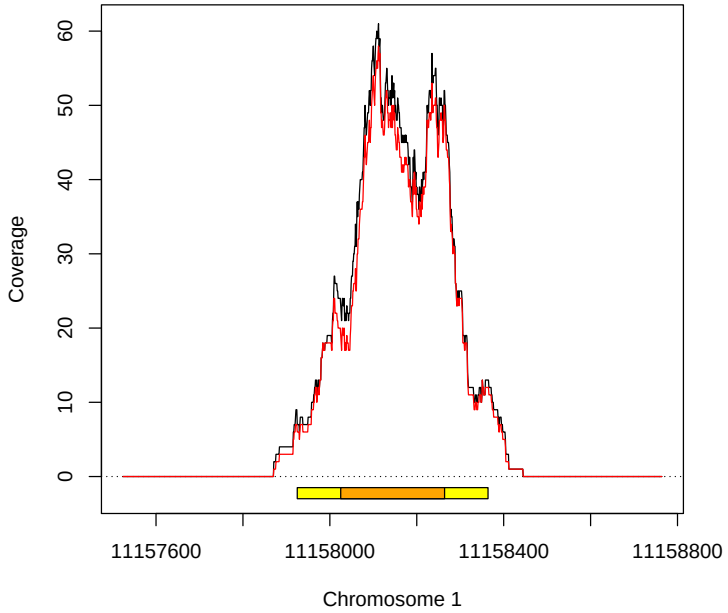

Figure 12: Coverage along chromosome plot for two samples.

LC_PAPER=en_US.UTF-8, LC_NAME=C, LC_ADDRESS=C, LC_TELEPHONE=C, LC_MEASUREMENT=en_US.UTF-8, LC_IDENTIFICATION=C

- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: IRanges 1.10.0, TEQC 1.0.0
- Loaded via a namespace (and not attached): Biobase 2.12.0, tools 2.13.0

9 References

Hummel M, Bonnin S, Lowy E, Roma G. TEQC: an R-package for quality control in target capture experiments. Bioinformatics 2011; doi: 10.1093/bioinformatics/btr122.

Bainbridge MN, Wang M, Burgess DL, Kovar C, Rodesch MJ, D'Ascenzo M, Kitzman J, Wu Y-Q, Newsham I, Richmond TA, Jeddeloh JA, Muzny D, Albert TJ, Gibbs RA. Whole exome capture in solution with 3 Gbp of data. Genome Biology 2010; 11:R62.

Gnirke A, Melnikov A, Maguire J, Rogov P, LeProust EM, Brockman W, Fennell T, Giannoukos G, Fisher S, Russ C, Gabriel S, Jaffe DB, Lander ES, Nusbaum C. Solution hybrid selection with ultra-long oligonucleotides for massively par-

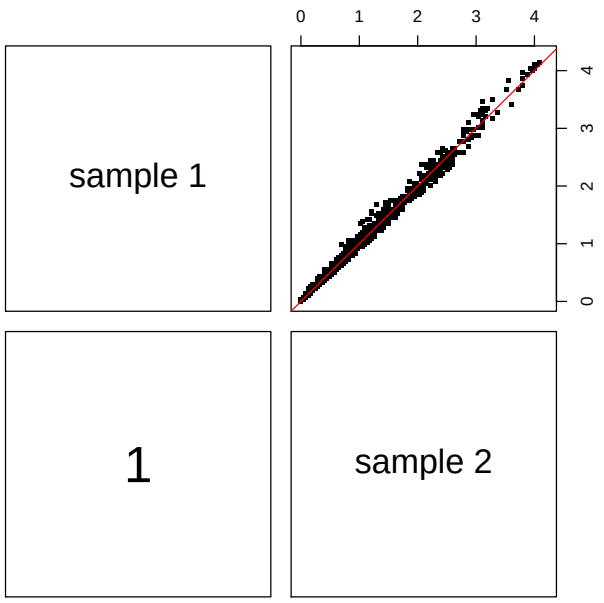

Figure 13: Correlation plots of normalized target coverage values of two samples.

allel targeted sequencing. Nat Biotechnol. 2009; 27(2): 182-9.

Tewhey R, Nakano M, Wang X, Pabon-Pena C, Novak B, Giuffre A, Lin E, Happe S, Roberts DN, LeProust EM, Topol EJ, Harismendy O, Frazer KA. Enrichment of sequencing targets from the human genome by solution hybridization. Genome Biol. 2009; 10(10): R116.