QoRTs Package User Manual

Stephen Hartley
National Human Genome Research Institute
National Institutes of Health

8 August 2014
Revised 8 August 2014
v0.0.3

Contents

1 Overview

The QoRTs software package is a fast, efficient, and portable toolkit designed primarily to aid in the detection and identification of errors, biases, and artifacts produced by paired-end high-throughput RNA-Seq technology. It can produce a wide variety of graphs, plots, and tables that allow the data to be visualized in various ways. Data can be compiled and contrasted in multiple ways to allow systematic errors or artifacts to reveal themselves more easily. While it will not directly assign pass/fail status, it is a powerful tool for bioinformaticians to detect and identify features in the data.

In (hopefully) most cases, these plots and graphs will not reveal anything other than mixed statistical noise. Next-Gen sequencing technologies have matured to the point where gross systematic errors and batch-specific biases are relatively modest and rare. However: mistakes can still occur, and basing conclusions on flawed data can be disastrous.

In addition to its primary function as a quality control tool, QoRTs simultaneously generates the feature-count files necessary for analysis with the *DESeq*/*DESeq2* [1] or *edgeR* [2] differential expression analysis tools, as well as the *DEXSeq* [3] differential-exon-usage analysis tool.

The QoRTs java utility is written in the Scala programming language (v2.11.1). However, it has been compiled to java byte-code and does not require an installation of Scala (or any other external libraries) in order to function. Thus, the entire QoRTs toolkit can be used in almost any operating system that supports java and R.

2 Requirements

The QoRTs software package comes in two distinct parts: a java jar-file and an R package. Installing the R package only requires R version 3.0.2 or higher. Running the java jar-file requires any version of java 6 or higher. The java utility does the bulk of the data processing, and will generally require at least 10-20gb of RAM. The R package is only responsible for some light data processing and for plotting/visualization, and thus has much lower resource requirements. It should run adequately on any reasonably-powerful 64-bit workstation.

QoRTs is designed to run on paired-end next-gen RNA-Seq data. The data must be aligned (or "mapped") to a reference genome before QoRTs can be run. RNA-Star [4], GSNAP [5], and TopHat2 [6] are all popular aligners for use with RNA-Seq data. The use of short-read or unspliced aligners such as BowTie, ELAND, BWA, or Novoalign is not recommended.

QoRTs requires that the bam files be sorted by read name. This can be accomplished via samtools using the command:

```
samtools sort -n unsorted.bam sorted
```

or via novosort, which may be faster and more memory-efficient:

```
novosort -n unsorted.bam > sorted.bam
```

In addition, QoRTs requires transcript annotations in the form of a gtf file. If you are using a annotation guided aligner it is likely you already have a transcript gtf file for your reference genome. We recommend you use the same annotation gtf for alignment, QC, and downstream analysis. We have found the Ensembl "Gene Sets" gtf[1] suitable for these purposes. However, any format that adheres to the gtf file specification[2] will work.

In general, it is assumed that the dataset consists of sequencing data derived from multiple biological samples, each with a known biological "condition". Using barcoding, it is possible to build a combined library of multiple distinct samples which can be run together on the sequencing machine and then demultiplexed afterward. This is often desirable, because it can be used to mitigate the effect of batch effects on downstream analysis. To produce sufficient data for downstream analysis, it may be necessary to run such combined libraries on multiple lanes. In this case, it is recommended that alignment and QoRTs QC analyses be performed BEFORE merging the data from these "technical replicates". This can make it easier to discern lane-specific sequencing artifacts from sample-specific abnormalities.

[1] Which can be acquired from the Ensembl website at `http://www.ensembl.org`

[2] See the gtf file specification at `http://genome.ucsc.edu/FAQ/FAQformat.html`

3 Preparations

The QoRTs package comes in two distinct parts: a java jar executable and an R package. First the java jar executable must be run on each bam file in the dataset. The bulk of the data processing takes place in this step. Second, the R package can be used to produce compiled graphs, plots, and tables.

3.1 Bam file decoder

Several QoRTs functions will require a "decoder" file, which describes each bam file. MORE INFO HERE!

Required Fields:

- *lanebam.ID*: The ID of the bam file. Must be unique.
- *lane.ID*: The ID of the lane or batch for the bam file.
- *group.ID*: The ID of the "group". For example: "Case" or "Control".
- *sample.ID*: The ID of the biological sample for the bam file. Each sample can have multiple bam-files associated with it.
- *qc.data.dir*: The directory in which the java utility should save all the output data.

Optional Fields: The following additional fields are optional. Note that you should not use these names

- *input.read.pair.count*: The number of reads in the original fastq file, before alignment.
- *multi.mapped.read.pair.count*: The number of reads that were multi-mapped by the aligner.
- *cycle.CT*: The number of bases in each read. By default this will be auto-detected from the data and need not be included.

In addition, the decoder can contain any other additional columns as desired, as long as all of the column names are distinct and do not match the column names listed above.

3.2 Example data

The separate R package *QoRTsExampleData* contains an example dataset with an example decoder:

```
directory <- system.file("extdata/", package="QoRTsExampleData",
                         mustWork=TRUE);
decoder.file <- system.file("extdata/decoder.txt",
                            package="QoRTsExampleData",
                            mustWork=TRUE);
decoder.data <- read.table(decoder.file,
                           header=T,
                           stringsAsFactors=F);
print(decoder.data);

##    lanebam.ID sample.ID lane.ID group.ID qc.data.dir
## 1    S_D1_RG1        D1      L1        D ex/S_D1_RG1
## 2    S_D1_RG2        D1      L2        D ex/S_D1_RG2
## 3    S_D1_RG3        D1      L3        D ex/S_D1_RG3
## 4    S_D7_RG1        D7      L1        D ex/S_D7_RG1
## 5    S_D7_RG2        D7      L2        D ex/S_D7_RG2
```

```
## 6    S_D7_RG3        D7      L3        D ex/S_D7_RG3
## 7    S_D9_RG1        D9      L1        D ex/S_D9_RG1
## 8    S_D9_RG2        D9      L2        D ex/S_D9_RG2
## 9    S_D9_RG3        D9      L3        D ex/S_D9_RG3
## 10   S_N4_RG1        N4      L1        N ex/S_N4_RG1
## 11   S_N4_RG2        N4      L2        N ex/S_N4_RG2
## 12   S_N4_RG3        N4      L3        N ex/S_N4_RG3
## 13   S_N6_RG1        N6      L1        N ex/S_N6_RG1
## 14   S_N6_RG2        N6      L2        N ex/S_N6_RG2
## 15   S_N6_RG3        N6      L3        N ex/S_N6_RG3
## 16   S_N8_RG1        N8      L1        N ex/S_N8_RG1
## 17   S_N8_RG2        N8      L2        N ex/S_N8_RG2
## 18   S_N8_RG3        N8      L3        N ex/S_N8_RG3
##    multi.mapped.read.pair.count input.read.pair.count
## 1                        445486               7742973
## 2                        448479               7821574
## 3                        465857               8105062
## 4                        403516               8127971
## 5                        408435               8221842
## 6                        427328               8584814
## 7                        368750               7720699
## 8                        371553               7800059
## 9                        390249               8129640
## 10                       363738               8034573
## 11                       367609               8118713
## 12                       384023               8471082
## 13                       472556               8678298
## 14                       478766               8767247
## 15                       539997               9802208
## 16                       387968               7391346
## 17                       393053               7475198
## 18                       411893               7792159
```

4 Processing of aligned RNA-Seq data

The first step is to process the aligned RNA-Seq data. The bulk of the data-processing is performed by the QoRTs.jar java utility. This tool produces an array of output files, analyzing and tabulating the data in various ways. This utility requires about 10-20gb of RAM for most genomes, and takes roughly 4-7 minutes to process 1 million read-pairs.

```
java -jar /path/to/jarfile/QoRTs.jar QC
                    mybamfile.bam
                    Rattus_norvegicus.RGSC3.4.69.gtf.gz
                    /qc/data/dir/path/
```

In the above command (which must be entered as a single line), you must replace `/path/to/jarfile/` with the file-path to the directory in which the jar file is kept. The path `/qc/data/dir/path/` should be replaced with the path you want the QC data to be written. This should match the path located in the decoder in the qc.data.dir column for this bam-file.

The bam processing tool includes numerous options. A full description of these options can be found by entering the command:

```
java -jar /path/to/jarfile/QoRTs.jar QC --man
```

There are a number of crucial points that require attention when using the QoRTs.jar QC command.

Stranded data: By default, QoRTs assumes that the data is NOT strand-specific. For strand-specific data, the `--stranded` flag must be added to the command line, like so:

```
java -jar /path/to/jarfile/QoRTs.jar QC
                    --stranded
                    mybamfile.bam
                    Rattus_norvegicus.RGSC3.4.69.gtf.gz
                    /qc/data/dir/path/
```

Stranded library type: The `--fr_secondStrand` option may be required depending on the stranded library type. QoRTs does not attempt to automatically detect the platform and protocol used for stranded data. There are two types of strand-specific protocols, which are described by the TopHat/CuffLinks documentation at `http://cufflinks.cbcb.umd.edu/manual.html#library` as `fr-firststrand` and `fr-secondstrand`. In HTSeq, these same library type options are defined as `-s reverse` and `-s yes` respectively. According to the CuffLinks manual, `fr-firststrand` (the default used by QoRTs for stranded data) applies to dUTP, NSR, and NNSR protocols, whereas `fr-secondstrand` applies to "Directional Illumina (ligation)" and "Standard SOLiD" protocols. If you are unsure which library type applies to your dataset, don't worry: one of the tests will report stranded library type. If you use this test to determine library type, be aware that you may have to re-run QoRTs with the correct library type set.

5 Visualization

Next, you must read in all the QC output from the java utility, using the command below. This command requires 2 arguments: a root directory and a decoder (which can be either a data frame or a file). We will be using the example data found in package *QoRTexampleData*, which is described in Section 3.2

```
res <- read.qc.results.data(directory, decoder.data.frame = decoder.data,
                            calc.DESeq2 = TRUE, calc.edgeR = TRUE);
```

Note that the `calc.DESeq2` and `calc.edgeR` options are optional, and tell QoRTs to attempt to load the *DESeq2* and *edgeR* packages respectively, and use the packages to calculate their respective normalization size factors. This is not strictly needed for most purposes, but allows QoRTs to plot the normalization factors against one another. See section 5.3.21 for more information.

5.1 Generating all default plots

To generate all the default compiled plots all at once, use the command:

```
makePlot.all.std(res,
             outfile.prefix = "./",
             plot.device.name = "CairoPNG");
```

This will usually take some time to run, but will produce all the compiled summary plots described in the rest of this section, including separate highlight plots for every sample in the dataset.

5.2 Plotting by sample, lane, or group

QoRTs includes automated methods for organizing and plotting the results in numerous different ways. The intent is to make patterns and biases more visible to the user.

All plotting functions in QoRTs require a `QoRT_Plotter` object. A `QoRT_Plotter` is a `RefClass` object that contains all the QC data along with a set of parameters that determine how to color and draw each bam file's data. A full accounting of all possible options available in the is beyond the scope of this manual, but can be found in the help docs for the `QoRT_Plotter` class.

5.2.1 Summary Plots

The simplest QoRTs plotter is the summary plotter. The summary plotter can be created using the command:

```
summary.plotter <- build.plotter.summary(res);
```

This plotter will plot all the bam-files on top of one another in semi-transparent gray. For example:

```
makePlot.insert.size(summary.plotter);
```

Which produces Figure 1:

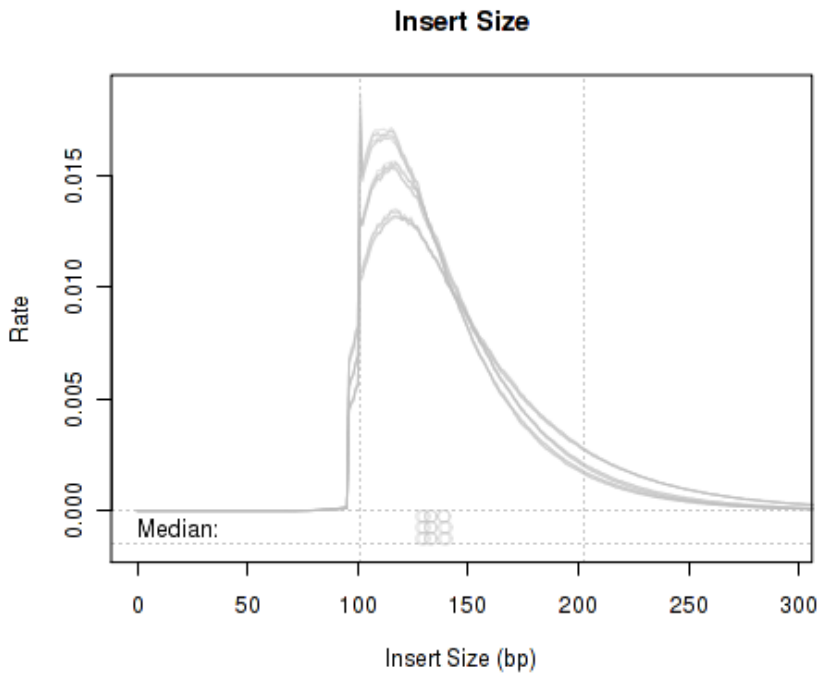

Figure 1: Phred Quality Score Plots

The above example plot displays the "Insert Size" of each bam file, as described in Section 5.3.5.

In addition, a compiled multi-plot in this style, containing all the standard QC plots, can be generated with the command:

```
makePlot.summary.basic(res);
```

Which produces Figure 2:

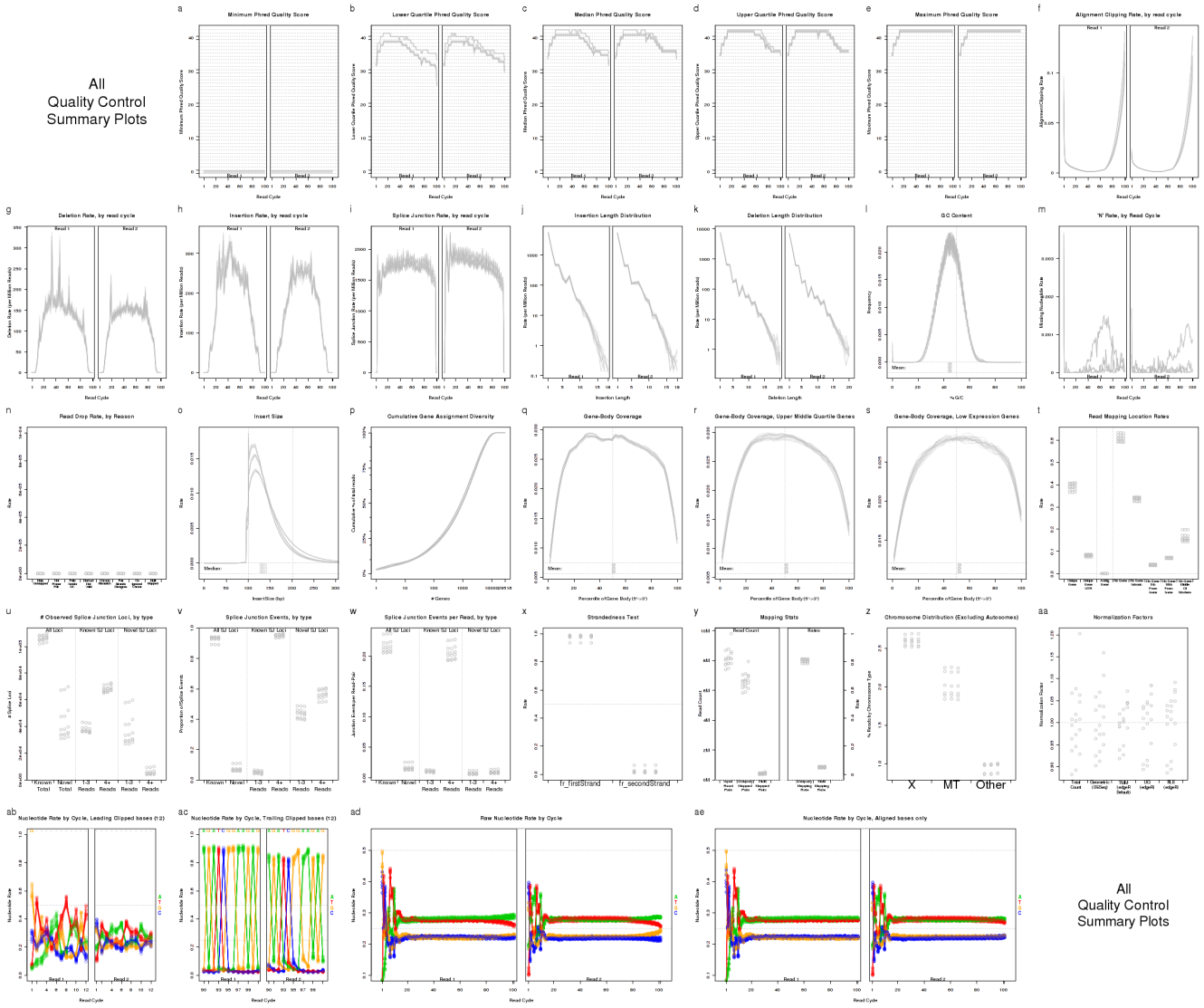

Figure 2: Compiled summary multi-plot

5.2.2 Colored by Lane/Batch

In order to more easily detect batch effects, it is possible to color each bam-file by lane/batch. For this, you make a plotter with the command:

```
byLane.plotter <- build.plotter.colorByLane(res);
```

The plotter will draw all the bam-files on top of one another, but color them based on their lane.ID. The plotter can then be used to create various QC plots, for example:

```
makePlot.insert.size(byLane.plotter);
makePlot.legend.over("topright",byLane.plotter);
```

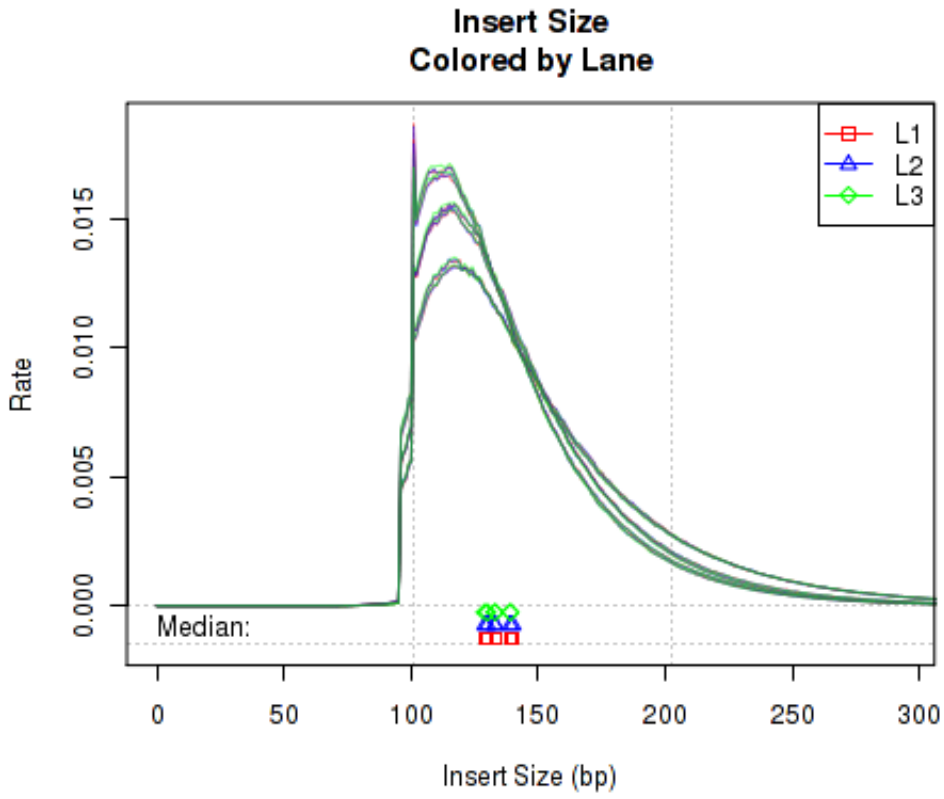

Figure 3: Phred Quality Score Plots

Which produces Figure 3:

The above example plot displays the "Insert Size" of each bam file, as described in Section 5.3.5.

In addition, a compiled multi-plot in this style, containing all the standard QC plots, can be generated with the command:

```
makePlot.summary.colorByLane(res);
```

5.2.3 Colored by Group/Phenotype

Next, it is sometimes useful to color samples by group.ID.

```
byGroup.plotter <- build.plotter.colorByGroup(res);
```

This plotter can then be used to create various QC plots, for example:

```
makePlot.insert.size(byGroup.plotter);
makePlot.legend.over("topright",byGroup.plotter);
```

Which produces Figure 4:

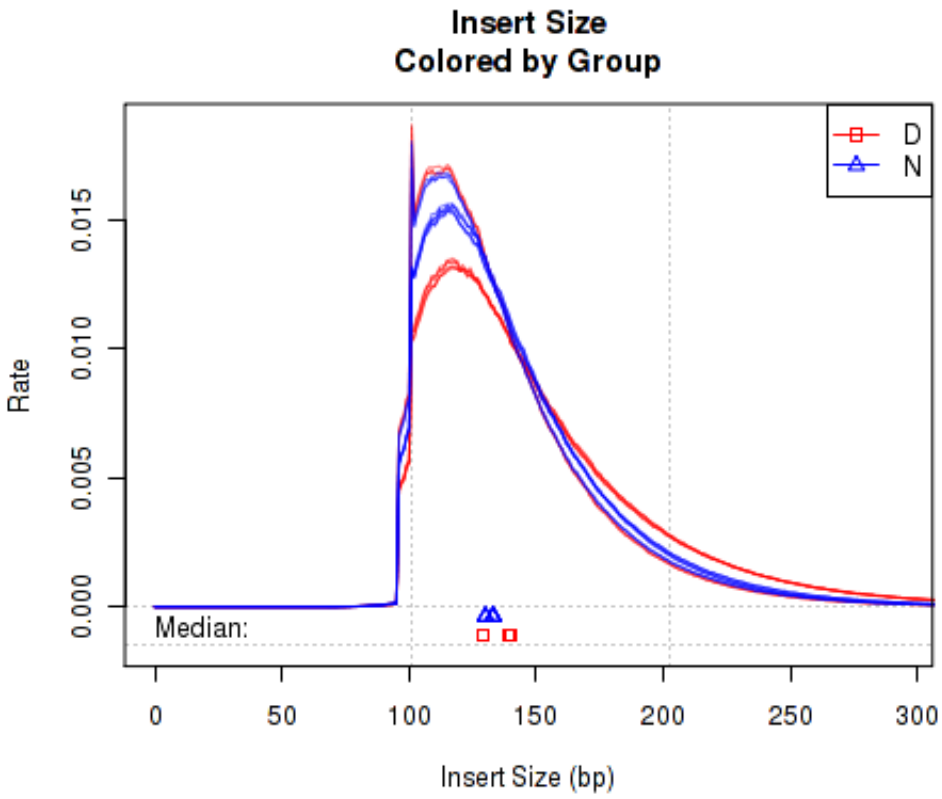

Figure 4: Phred Quality Score Plots

The above example plot displays the "Insert Size" of each bam file, as described in Section 5.3.5.

In addition, a compiled multi-plot in this style, containing all the standard QC plots, can be generated with the command:

```
makePlot.summary.colorByGroup(res);
```

5.2.4 Basic Sample Highlight

Sometimes it is useful to highlight an individual sample.

```
sample.D1.plotter <- build.plotter.highlightBySample("D1",res);
```

This plotter can then be used to create various QC plots, for example:

```
makePlot.insert.size(sample.D1.plotter);
makePlot.legend.over("topright",sample.D1.plotter);
```

Which produces Figure 5:

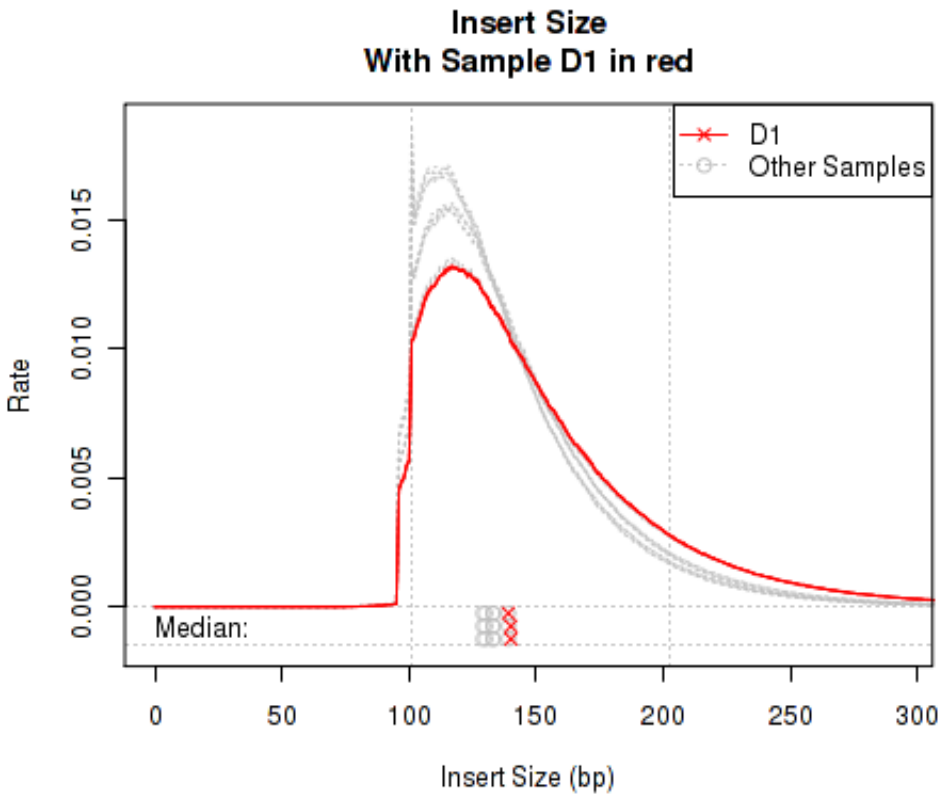

Figure 5: Phred Quality Score Plots

The above example plot displays the "Insert Size" of each bam file, as described in Section 5.3.5.

In addition, a compiled multi-plot in this style, containing all the standard QC plots, can be generated with the command:

```
makePlot.summary.sample.highlight(res,
                    curr.sample = "D1");
```

5.2.5 Sample Highlight, Colored by Lane

Sometimes it can be useful to highlight an individual sample. However, if that sample has multiple bam files ("technical replicates", derived from multiple separate lanes/runs on the same library), it can be useful to color the different runs with different distinct colors. With this plotter, only the "highlighted" sample is colored, all other samples are colored Gray.

```
sample.D1.colorByLane.plotter <-
            build.plotter.highlightBySample.colorByLane("D1",res);
```

This plotter can then be used to create various QC plots, for example:

```
makePlot.insert.size(sample.D1.colorByLane.plotter);
makePlot.legend.over("topright",sample.D1.colorByLane.plotter);
```

Which produces Figure 6:

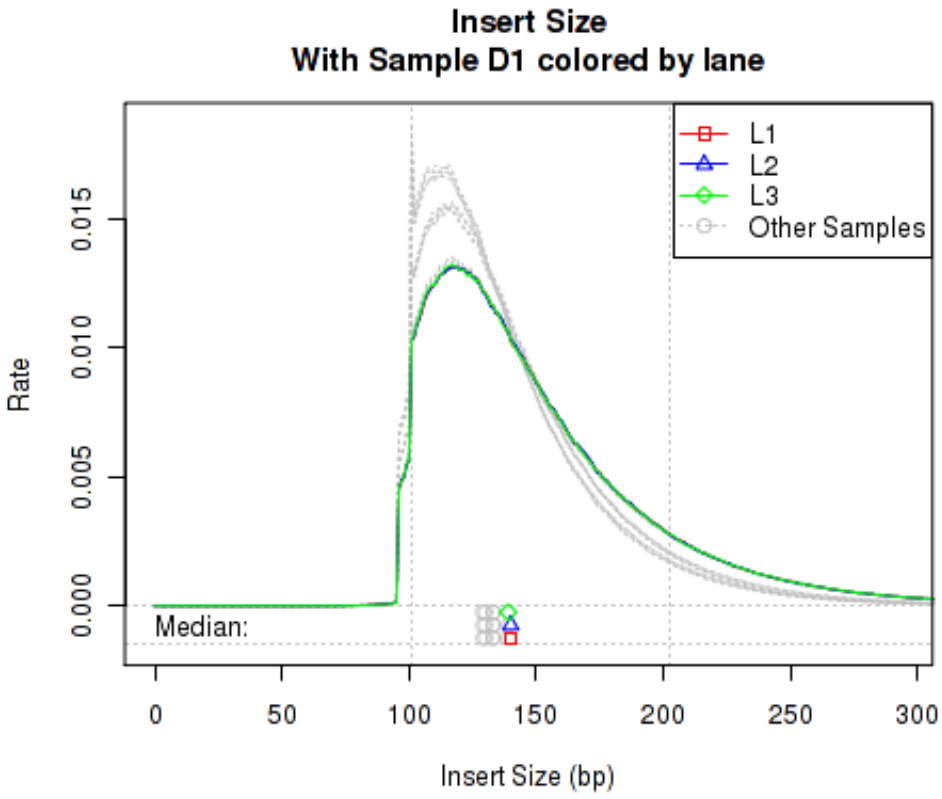

Figure 6: Phred Quality Score Plots

The above example plot displays the "Insert Size" of each bam file, as described in Section 5.3.5.

In addition, a compiled multi-plot in this style, containing all the standard QC plots, can be generated with the command:

```
makePlot.summary.sample.highlight.andColorByLane(res,
                    curr.sample = "D1");
```

5.3 Description of Individual Plots

QoRTs is capable of producing a wide variety of different plots and graphs. While most of these plots will not be particularly interesting or informative in the majority of cases, they may reveal artifacts or errors if and when they occur.

The example plots in the following section all use the `byLane.plotter` object (from Section 5.2.2), which colors each bamfile by its lane ID.

5.3.1 Phred Quality Score

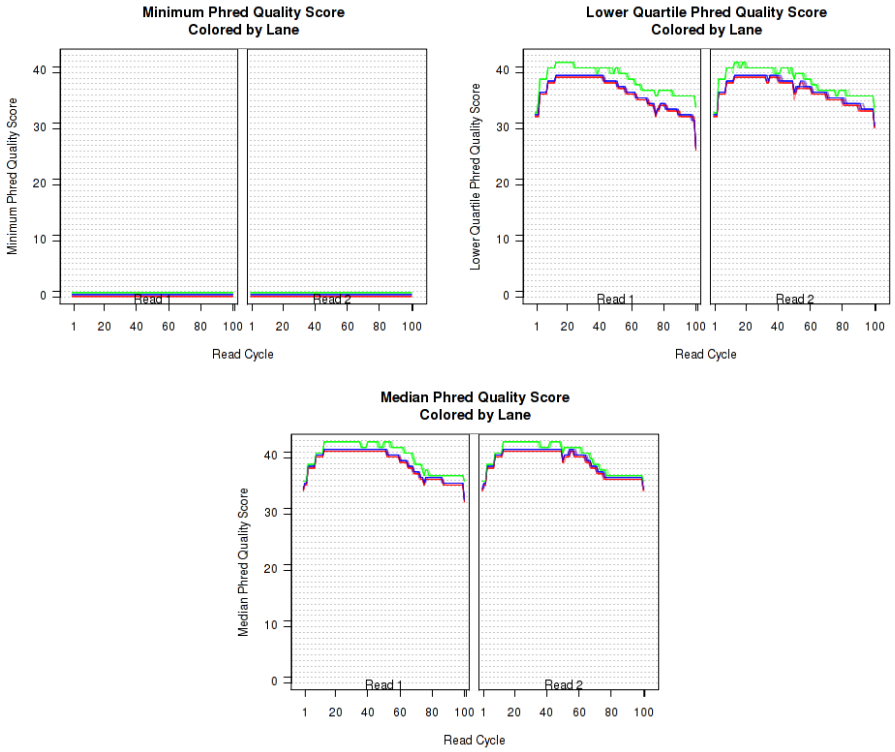

Figure 7: Phred Quality Score Plots

The plots shown in Figure 7 displays information about the phred quality score (y-axis) as a function of the position in the read (x-axis). Five statistics can be plotted: minimum, maximum, upper and lower quartiles, and median. These statistics are calculated individually for each bam file and each read position (ie, each plotted line corresponds to a bam file).

Note that the Phred score is always an integer, and as such these plots would normally be very difficult to read because lines would be plotted directly on top of one another. To reduce this problem, the plots are vertically offset from one another.

These plots can be generated individually with the commands:

```
makePlot.qual.pair(byLane.plotter,"min");
makePlot.qual.pair(byLane.plotter,"lowerQuartile");
makePlot.qual.pair(byLane.plotter,"median");
```

Additional options (Not shown):

```
makePlot.qual.pair(byLane.plotter,"upperQuartile");
makePlot.qual.pair(byLane.plotter,"max");
```

5.3.2 GC Content

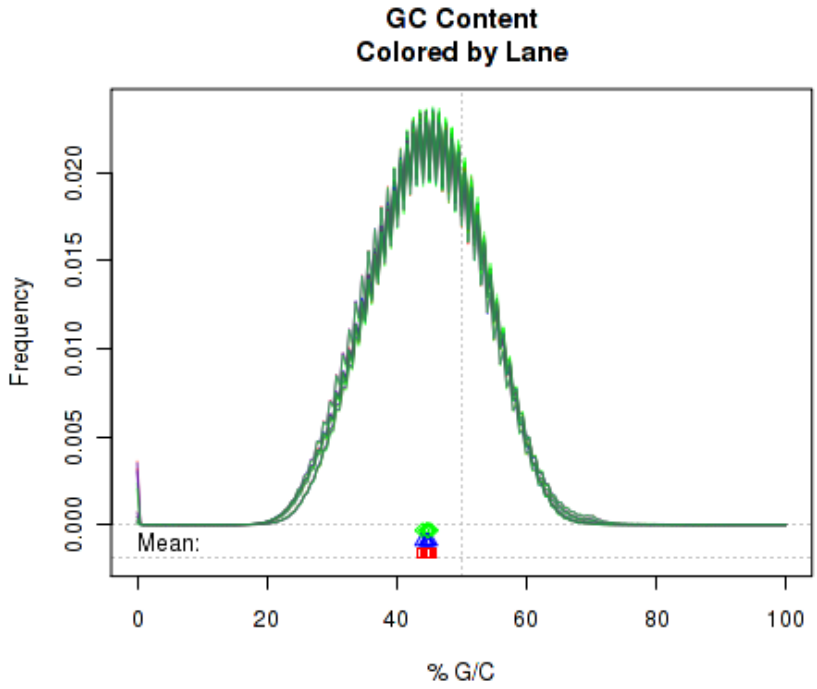

Figure 8: GC Bias

For each bam file, Figure 8 displays a histogram showing the frequency that different proportions of G and C (versus A, T, and N) appear in the bam-file's reads. Each plotted line corresponds to a bam-file. At the bottom of the plot the mean average G/C content is also plotted. Once again, the means are offset from one another by lane, to allow for easy detection of batch effects.

Note on the example data: You may note that the GC-content plot in the example dataset shows alternating spikes. This is due to the fact that the GC-content is calculated for each read-pair. As such, the number of G's and C's will always be even in the subset of read-pairs that are fully overlapping. In this dataset this is even true when the read-pairs are more than fully overlapped, since the initial adapter sequence is identical on both ends.

This plot can be generated individually with the command:

```
makePlot.gc(byLane.plotter);
```

5.3.3 Clipping Profile

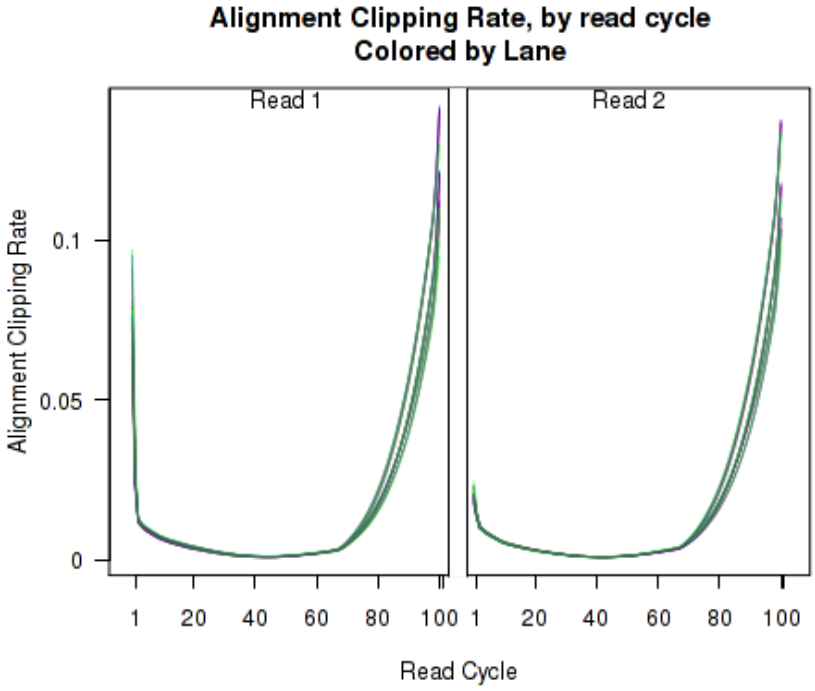

Figure 9: Clipping Profile

For each bam file, Figure 9 displays the rate (y-axis) at which the aligner soft-clips the reads as a function of read position (x-axis). Note that this will only be informative when using aligners that are capable of soft-clipped alignment (such as RNA-Star or GSNAP, but *not* TopHat).

This plot can be generated individually with the command:

```
makePlot.clipping(byLane.plotter);
```

5.3.4 Cigar Op Profile

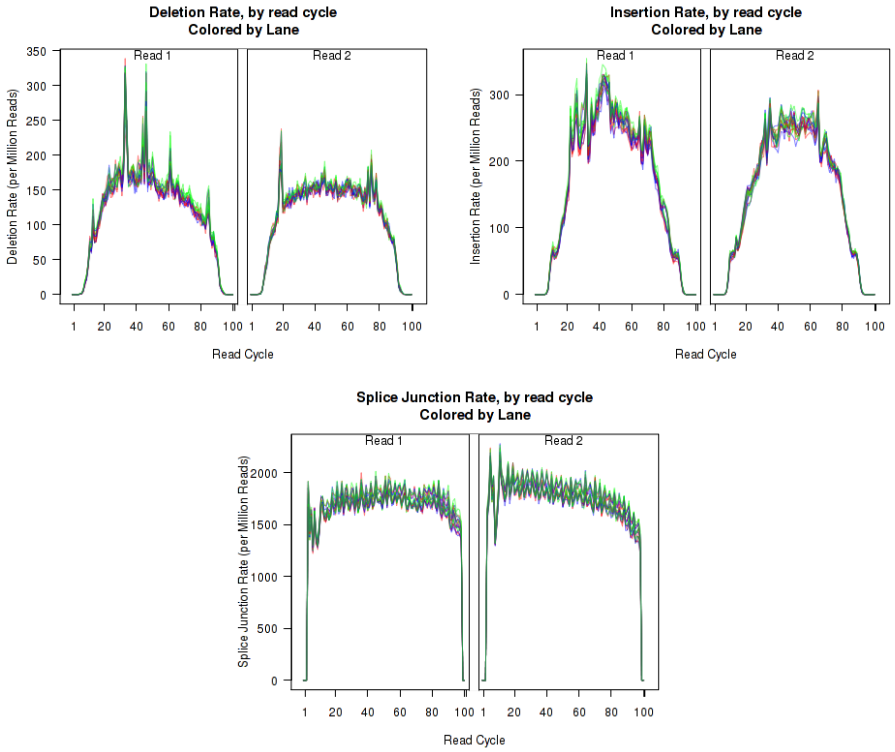

Figure 10: Cigar Operation Profiles

For each bam file, Figure 10 displays the rate (y-axis) of various cigar operations as a function of read position (x-axis). All 9 legal cigar operations can be plotted, but for most purposes only Deletions, Insertions, and Splice junctions will be informative.

This plot can be generated with the command:

```
makePlot.cigarOp.byCycle(byLane.plotter, "Del");
makePlot.cigarOp.byCycle(byLane.plotter, "Ins");
makePlot.cigarOp.byCycle(byLane.plotter, "Splice");
```

5.3.5 Insert Size

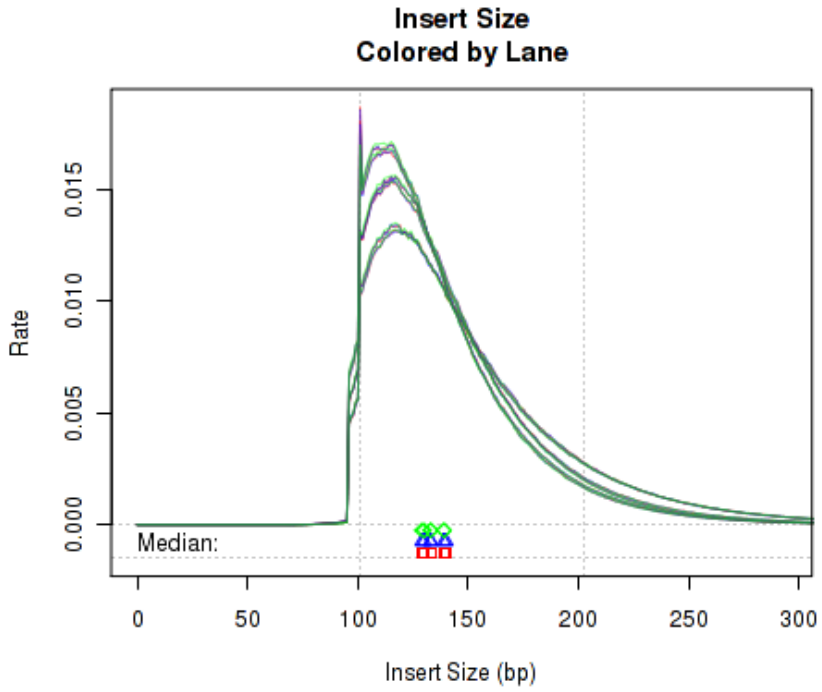

Figure 11: Insert Size

For each bam file, Figure 11 displays a histogram of the "insert size". Each line corresponds to one bam-file, and displays the rate (y-axis) at which that bam-file's reads possess a given insert size (x-axis).

Definition: "Insert Size": The "insert size" is the length (in base-pairs) between the two sequencing adapters for a pair of paired-end reads. In other words, it is the size of the original RNA fragment.

Insert Size Estimation: The Insert size is calculated using the alignment of the paired reads. When the two paired reads are aligned such that they overlap with one another the insert size can be calculated exactly. In such cases, the calculation of the insert size does *not* depend on the transcript annotation. However, when there is no overlap the exact insert size can be uncertain. Multiple splice junctions may lie in the region between the endpoints of the two paired reads, and there is no real way to determine which junctions the fragment used, if any. QoRTs uses the set of all splice junctions found between the endpoints of the two reads, and uses the shortest possible path from endpoint to endpoint. In some cases this may *under-estimate* the insert size, as the actual path may not be the shortest possible path. In other cases this may also *over-estimate* the insert size, if the RNA fragment includes novel splice junctions not found in the transcript annotation. However, in most cases this method appears to produce a reasonably good approximation of the insert size.

Note that the median average insert sizes for each bam-file are plotted below the main plot. Each point corresponds to one bam-file.

This plot can be generated individually with the command:

```
makePlot.insert.size(byLane.plotter);
```

5.3.6 Gene-Body Coverage

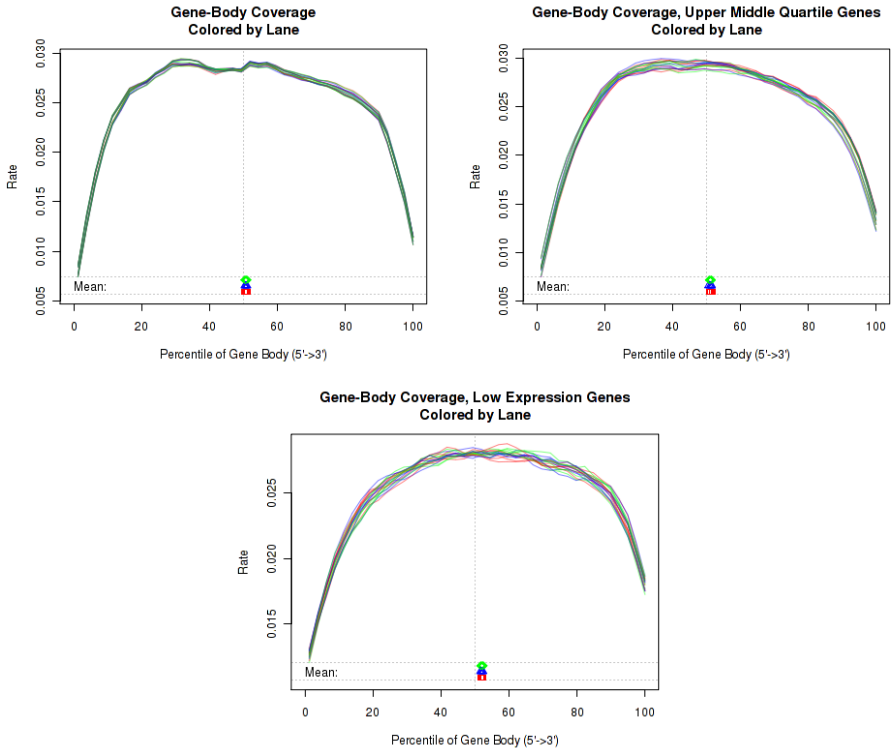

Figure 12: Gene-Body Coverage

For each bam-file, the leftmost plot of Figure 12 displays the coverage profile across quantiles of all genes' lengths, from 5' to 3'. The middle plot displays the coverage profile for only the genes that are in the upper-middle quartile by read-count. The leftmost plot displays the coverage profile for the genes that are in the two lower quartiles.

Minor notes: To calculate the coverage profile, all the transcripts for each gene are merged together into a single "flat" pseudo-transcript which contains all exonic regions belonging to the gene. For each gene, the pseudo-transcript is broken up into 40 equal-length counting bins, so that each bin contains 2.5% of the total gene length. Each read-pair is counted once for every counting bin with which it overlaps. Any genes that overlap with other genes are automatically excluded. Any reads that overlap with more than one gene are automatically excluded. Any genes that have zero reads on a given bam file are automatically excluded for the purposes of finding the gene quantiles.

This plot can be generated individually with the command:

```
makePlot.genebody.coverage(byLane.plotter);
makePlot.genebody.coverage.UMQuartile(byLane.plotter);
makePlot.genebody.coverage.lowExpress(byLane.plotter);
```

5.3.7 N-Rate

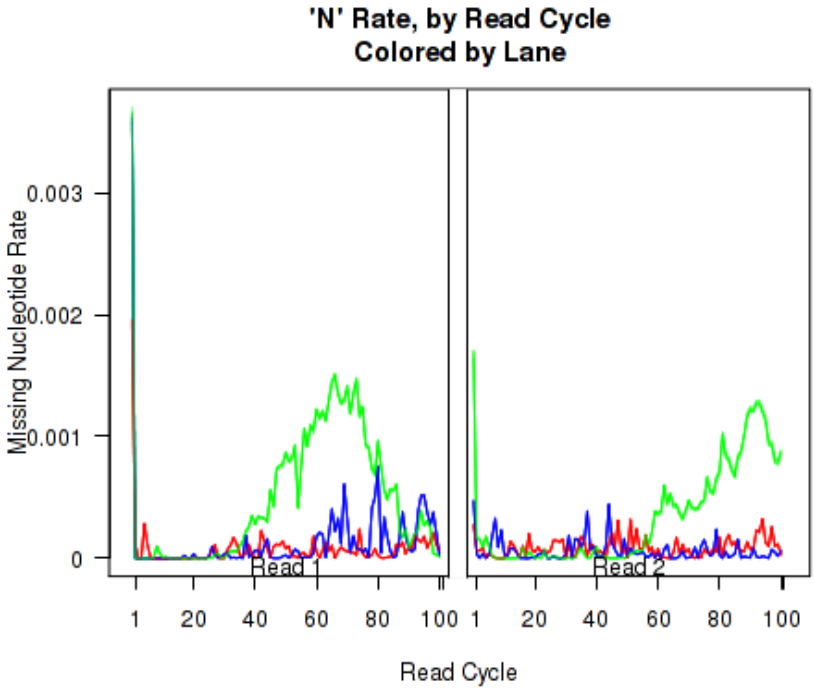

Figure 13: PLOT NAME

Figure 13 displays the rate (y-axis) at which the read sequence is "N" (or "missing"), as a function of the read position (x-axis). Each line corresponds to one bam-file.

This plot can be generated individually with the command:

```
makePlot.missingness.rate(byLane.plotter);
```

5.3.8 Cigar Length Distribution

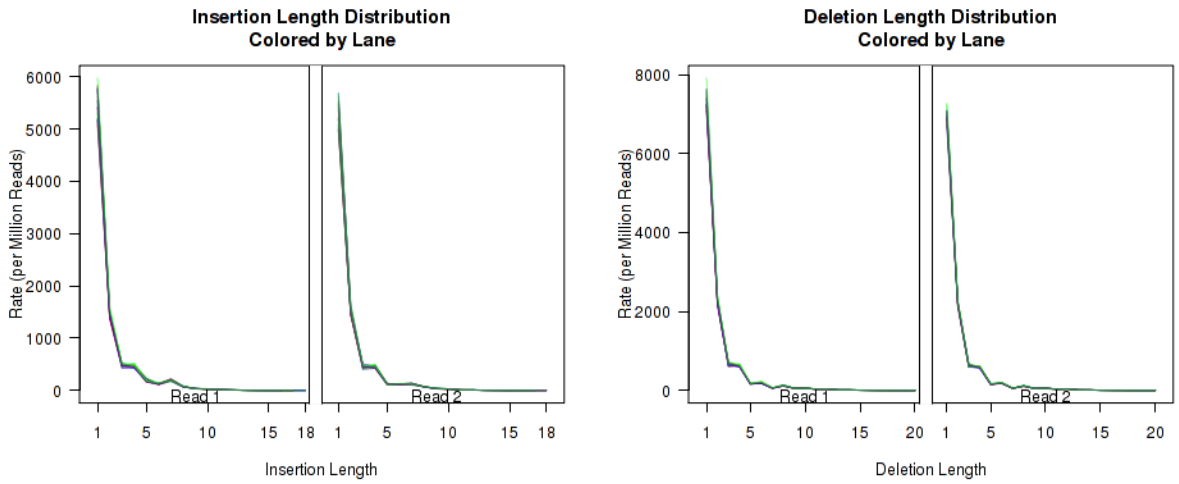

Figure 14: Cigar Length Distribution

The plots in Figure 14 display histograms of cigar operation length.
These plots can be generated individually with the commands:

```
makePlot.cigarLength.distribution(byLane.plotter, "Ins");
makePlot.cigarLength.distribution(byLane.plotter, "Del")
```

5.3.9 Cumulative Gene Diversity

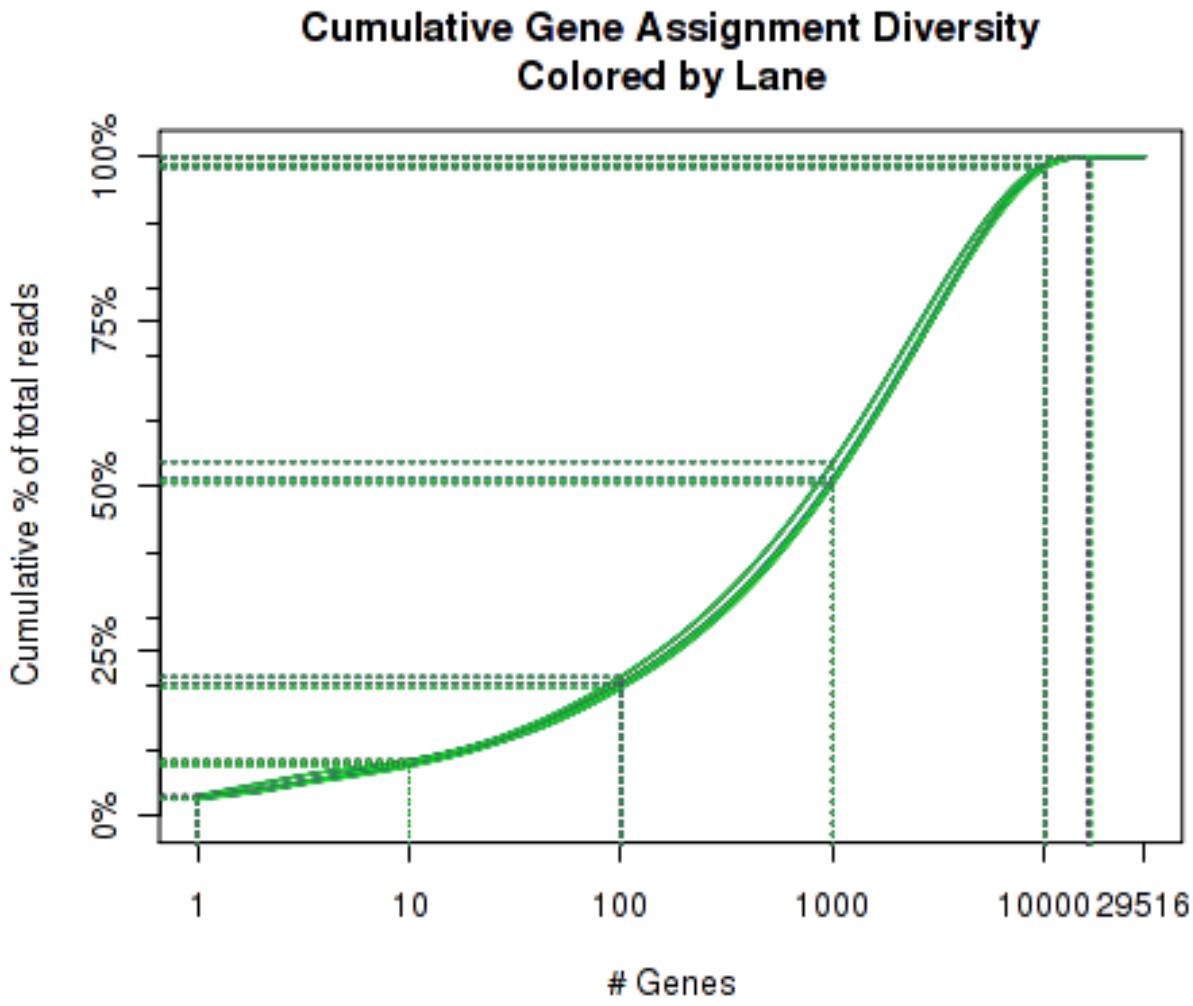

Figure 15: Cumulative Gene Diversity

For each bam-file, Figure 15 displays the cumulative gene diversity. For each bam-file, the genes are sorted by read-count. Then, a cumulative function is calculated for the percent of the total proportion of reads as a function of the number of genes. Intercepts are plotted as well, showing the cumulative percent for 1 gene, 10 genes, 100 genes, 1000 genes, and 10000 genes.

So, for example, across all the bam-files, around 50 to 55 percent of the read-pairs were found to map to the top 1000 genes. Around 20 percent of the reads were found in the top 100 genes. And so on.

This can be used as an indicator of whether a large proportion of the reads stem from of a small number of genes.

Note that this is restricted to only the reads that map to a single unique gene. Reads that map to more than one gene, or that map to intronic or intergenic areas are ignored.

This plot can be generated individually with the command:

```
makePlot.gene.cdf(byLane.plotter);
```

5.3.10 Nucleotide Rates, by Cycle

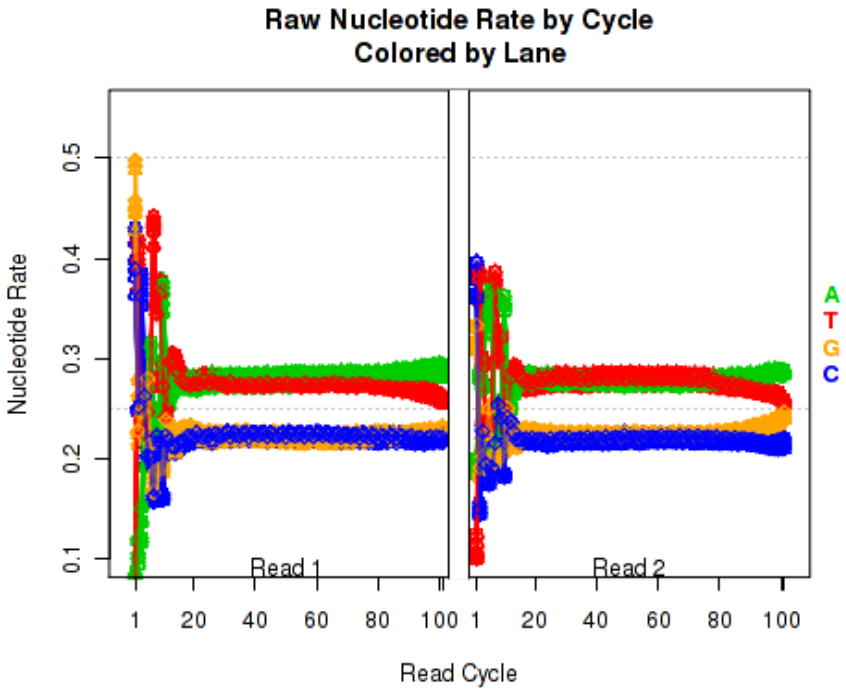

Figure 16: Nucleotide rates, by cycle

For each bam-file, Figure 16 displays the rate at which each nucleotide appears (y-axis), as a function of the position in the read (x-axis). For each plot, each bam-file has 4 lines associated with it, one for each nucleotide. Note that, like all the other nucleotide-rate plots, this plot does not use the same convention for coloring as the other plots: in these plots the line coloration indicates the nucleotide.

This plot displays the "raw" nucleotide rates, including bases that are soft-clipped by the aligner.

This plot can be generated individually with the command:

```
makePlot.raw.NVC(byLane.plotter);
```

5.3.11 Aligned Nucleotide Rates, by Cycle

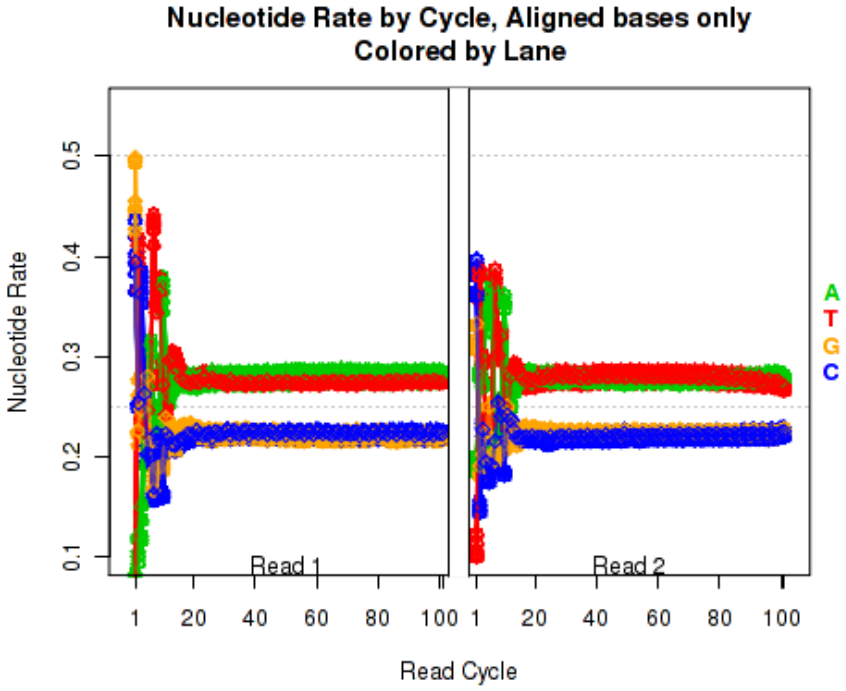

Figure 17: Aligned nucleotide rates, by cycle

Figure 17 is identical to Figure 16 (described in section 5.3.10), except that it only counts bases that are *not* soft clipped off by the aligner.

This plot can be generated individually with the command:

```
makePlot.minus.clipping.NVC(byLane.plotter);
```

5.3.12 Leading Clipped Nucleotide Rates

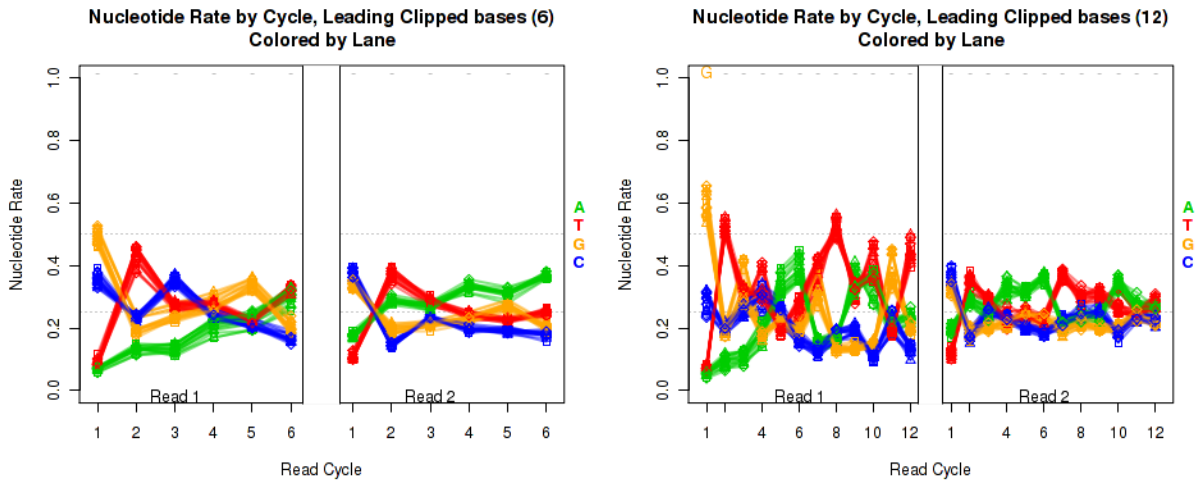

Figure 18: Leading-clipped nucleotide rates

The left plot in Figure 18 displays the nucleotide rate (y-axis) as a function of read position (x-axis), for the first 6 bases of reads in which *exactly* 6 bases were clipped off the 5' end. The right plot displays the nucleotide rate (y-axis) as a function of read position (x-axis), for the first 12 bases of reads in which *exactly* 12 bases were clipped off the 5' end.

This plot can be generated individually with the command:

```
makePlot.NVC.lead.clip(byLane.plotter, clip.amt = 6);
makePlot.NVC.lead.clip(byLane.plotter, clip.amt = 12);
```

Any integer can be used as the `clip.amt` value.

5.3.13 Trailing Clipped Nucleotide Rates

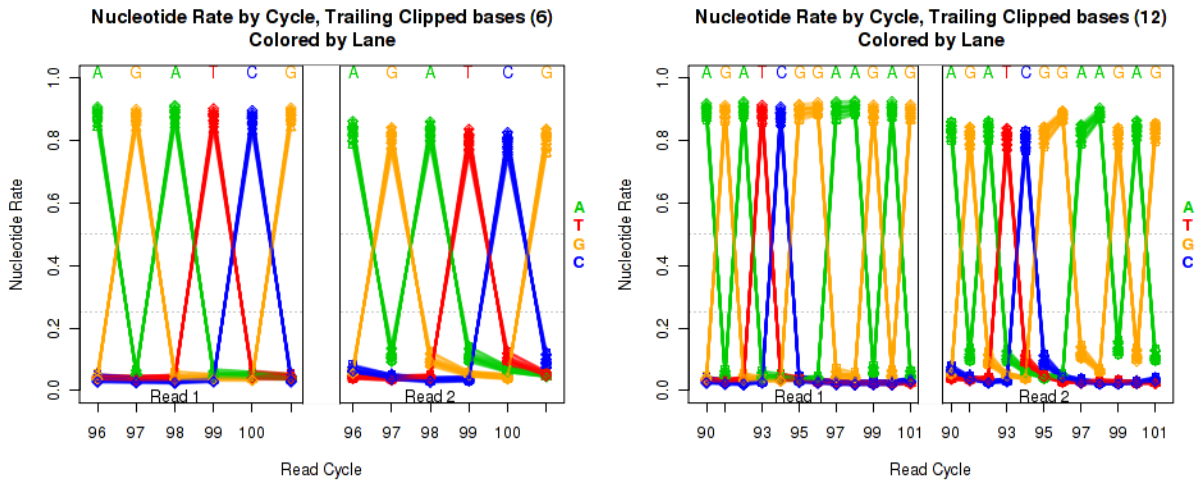

Figure 19: Trailing-clipped nucleotide rates

The left plot in Figure 19 displays the nucleotide rate (y-axis) as a function of read position (x-axis), for the last 6 bases of reads in which *exactly* 6 bases were clipped off the 3' end. The right plot displays the nucleotide rate (y-axis) as a function of read position (x-axis), for the last 12 bases of reads in which *exactly* 12 bases were clipped off the 3' end.

Note concerning the example data: In the example dataset an extremely strong trend is easily visible. The specific sequence observed matches that of the sequencing adapter used. The pattern appears in reads coming from fragments that are smaller than the read length. In these cases, the 3' end of each read will continue into the adapter sequence after sequencing the entire template fragment. Thus: for the left and right plots the sequence comes from reads with an insert size of exactly 95 and 89, respectively (ie 101 base pairs minus 6 or 12).

These plots can be generated individually with the command:

```
makePlot.NVC.tail.clip(byLane.plotter, clip.amt = 6);
makePlot.NVC.tail.clip(byLane.plotter, clip.amt = 12);
```

Any integer can be used as the `clip.amt` value.

5.3.14 Mapping location rates

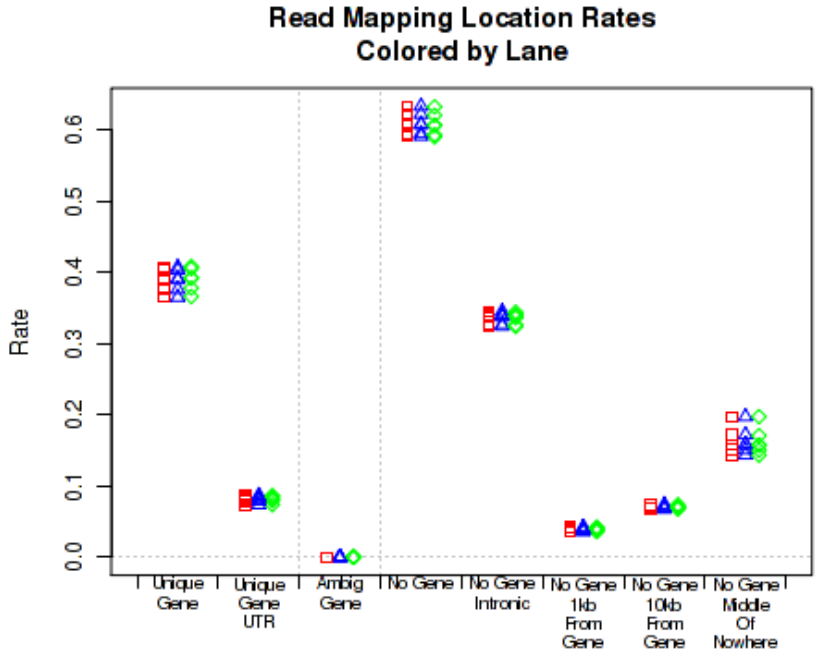

Figure 20: Gene assignment rates

For each bam-file, Figure 20 this plot displays the rate (y-axis) for which the bam-file's read-pairs are assigned to the given categories.
The categories are:

- *Unique Gene*: The read-pair overlaps with the exonic segments of one and only one gene. For many downstream analyses tools, such as DESeq, DESeq2 [1] and *EdgeR* [2], only read-pairs in this category are used.
- *Ambig Gene*: The read-pair overlaps with the exons of more than one gene.
- *No Gene*: The read-pair does not overlap with the exons of any annotated gene.
- *No Gene, Intronic*: The read-pair does not overlap with the exons of any annotated gene, but appears in a region that is bridged by an annotated splice junction.
- *No Gene, 1kb from gene*: The read-pair does not overlap with the exons of any annotated gene, but is within 1 kilobase from the nearest annotated gene.
- *No Gene, 10kb from gene*: The read-pair does not overlap with the exons of any annotated gene, but is within 10 kilobases from the nearest annotated gene.
- *No Gene, middle of nowhere*: The read-pair does not overlap with the exons of any annotated gene, and is *more* than 10 kilobases from the nearest annotated gene.

This plot can be generated individually with the command:

```
makePlot.gene.assignment.rates(byLane.plotter);
```

5.3.15 Splice Junction Loci

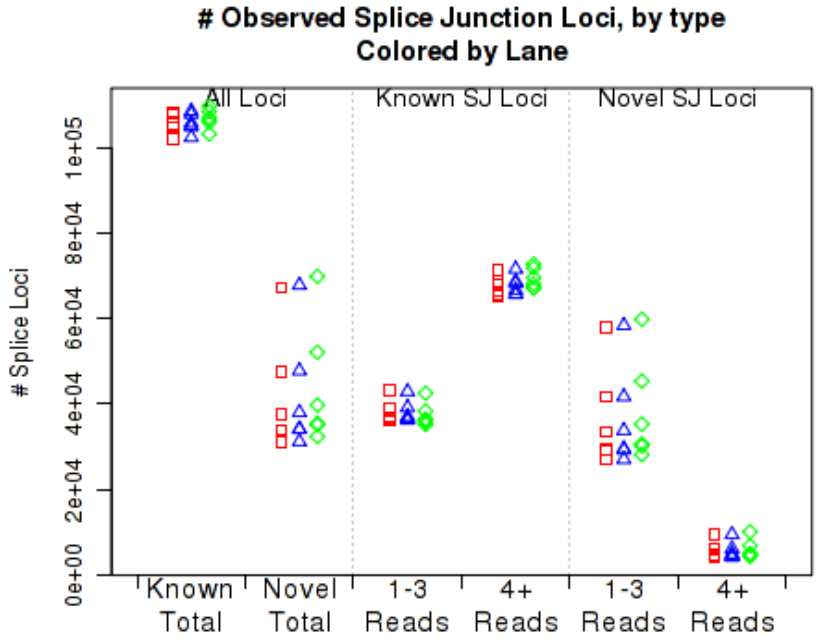

Figure 21: Splice junction loci

For each bam-file, Figure 21 displays the number (y-axis) of splice junction *loci* of each type that appear in the bam-file's reads. Splice junctions are split into 4 groups, first by whether the splice junction appears in the transcript annotation gtf ("known" vs "novel"), and then by whether the splice junction has 4 or more reads covering it, or 1-3 reads ("Hi" vs "Lo").

This plot can be used to detect a number of anomalies. For example: whether mapping or sequencing artifacts caused a disproportionate discovery of novel splice junctions in one sample or batch.

This plot can be generated individually with the command:

```
makePlot.splice.junction.loci.counts(byLane.plotter);
```

5.3.16 Splice Junction Event Rates

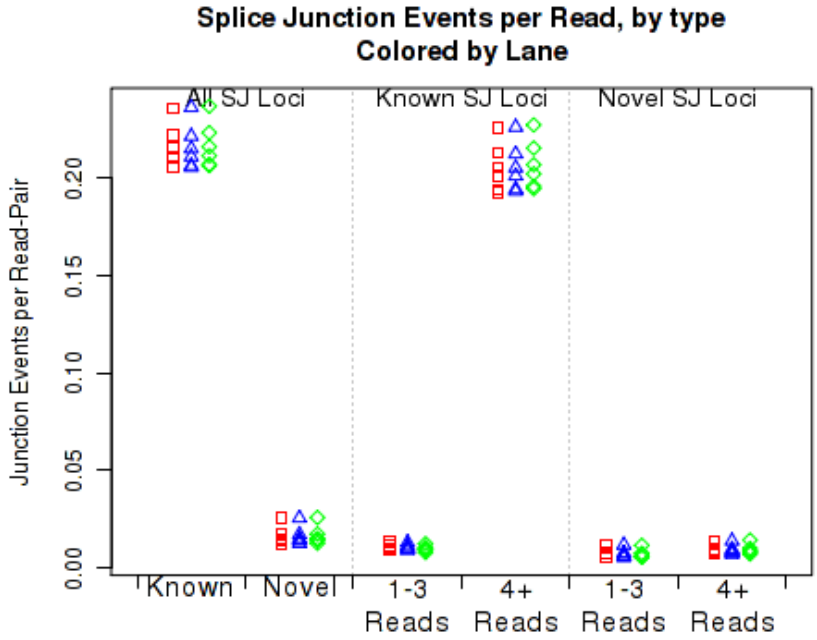

Figure 22: Splice junction events

For each bam-file, Figure 22 displays the rate (y-axis) at which each type of splice junction appear in the bam-file's reads. Splice junctions are split into 4 groups, first by whether the splice junction appears in the transcript annotation gtf ("known" vs "novel"), and then by whether the splice junction has 4 or more reads covering it, or 1-3 reads ("Hi" vs "Lo").

This plot is used to detect whether sample-specific or batch effects have a substantial or biased effect on splice junction appearance, either due to differences in the original RNA, or due to artifacts that alter the rate at which the aligner maps across splice junctions.

This plot can be generated individually with the command:

```
makePlot.splice.junction.event.rates(byLane.plotter);
```

5.3.17 Splice Junction Event type

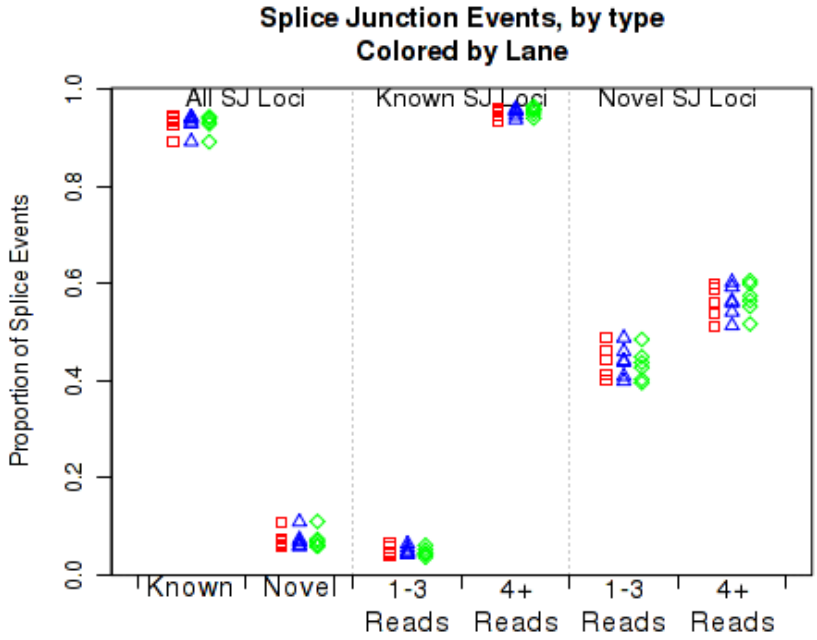

Figure 23: Splice junction events

For each bam-file, Figure 23 displays the rates (y-axis) at which splice junctions appear in each bam-file's reads. The first two columns display the overall rate at which splice junction events are on known (eg, annotated) splice loci versus novel loci. The other four columns display the rates at which these events occur on high-coverage loci vs low-coverage loci for known and novel junction types.

This plot is used to detect whether sample-specific or batch effects have a substantial or biased effect on splice junction appearance, either due to differences in the original RNA, or due to artifacts that alter the rate at which the aligner maps across splice junctions.

This plot can be generated individually with the command:

```
makePlot.splice.junction.event.rates.split.by.type(byLane.plotter);
```

5.3.18 Strandedness test

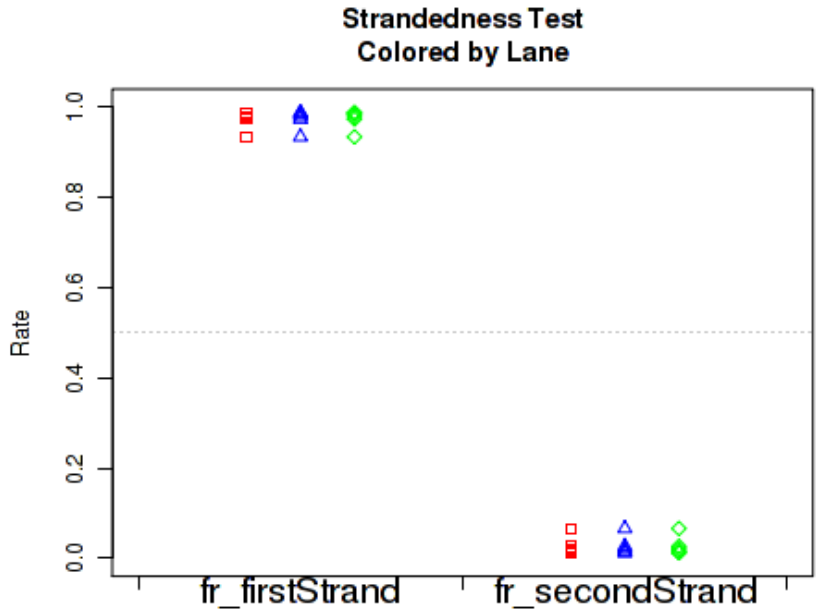

Figure 24: Strandedness

Figure 24 displays the rate at which reads appear to follow the two possible library-type strandedness rules. (See section 4 for more information on stranded library types).

This plot is used to detect whether your data is indeed stranded, and whether you are using the correct stranded data library type option. For unstranded libraries, one would expect all points to fall very close to the 50-50 center line. For stranded libraries, all points should fall closer to 99

If (and only if) all the bam files were run using the same strandedness and library type options, then green target boxes will be drawn around the areas where the points should appear. If points appear substantially outside these boxes, then you may be running QoRTs using the wrong library type options.

This plot can be generated individually with the command:

```
makePlot.strandedness.test(byLane.plotter);
```

5.3.19 Mapping stats

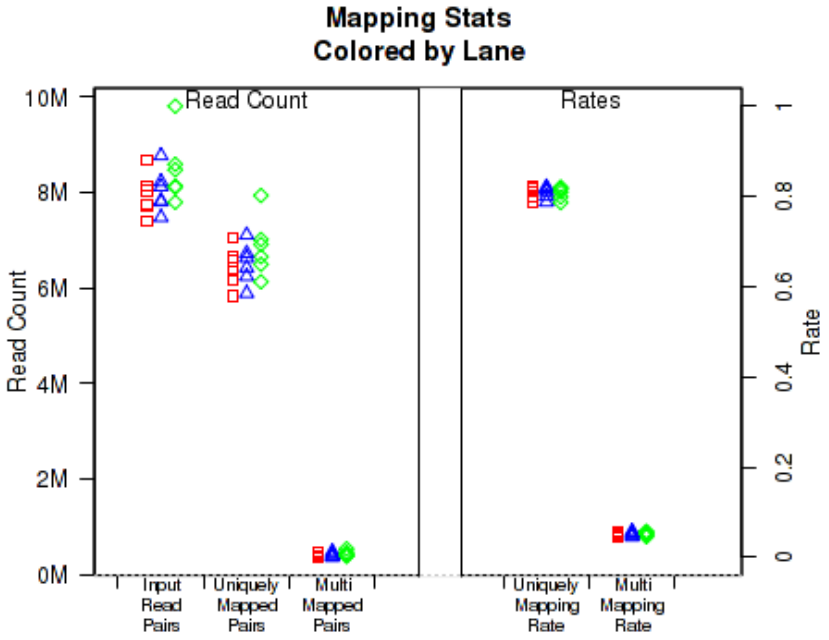

Figure 25: Mapping stats

For each bam file, Figure 25 displays the mapping rates and counts.
This plot can be generated individually with the command:

```
makePlot.mapping.rates(byLane.plotter);
```

5.3.20 Chromosome counts

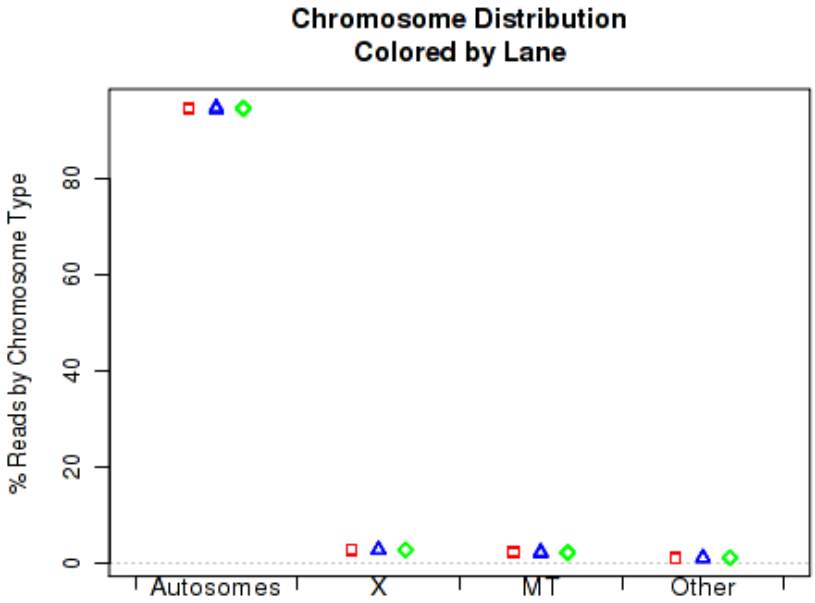

Figure 26: Mapping stats

For each bam file, Figure 26 displays the number of read-pairs mapping to each category of chromosome.

The `chromosome.name.style` must be set to match the style of your chromosome names. By default it assumes the chromosomes are named chr1, chr2, chr3, etc.

For more information, see the help document using the command `help(makePlot.chrom.type.rates)`.

This plot can be generated individually with the command:

```
makePlot.chrom.type.rates(byLane.plotter);
```

5.3.21 Normalization Factors

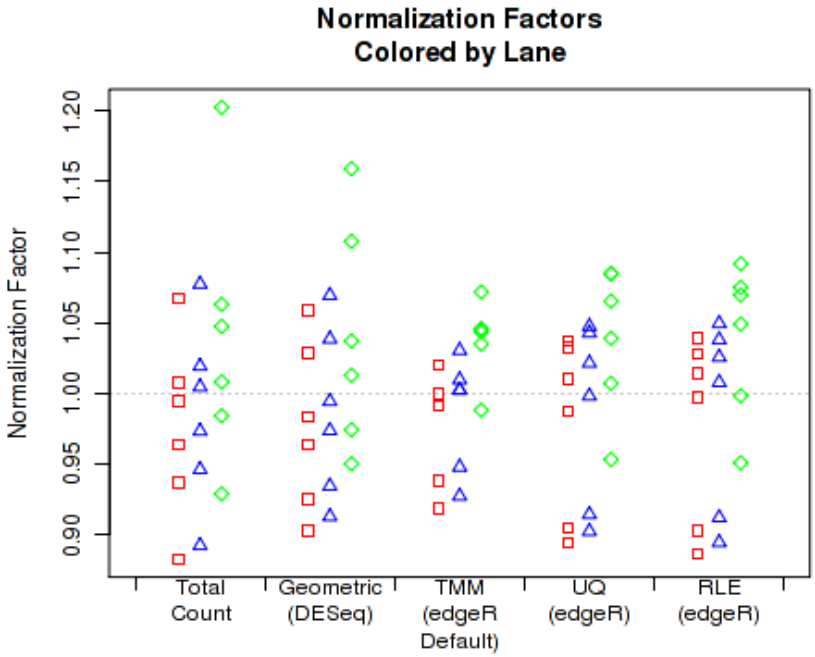

Figure 27: Mapping stats

For each bam file, Figure 27 displays the normalization factors.

By default, QoRTs will automatically detect whether *DESeq2* and *edgeR* are installed and will use these tools to calculate their respective normalization size factors. If neither package is found, then it will only plot the total count normalization.

This plot can be generated individually with the command:

```
makePlot.norm.factors(byLane.plotter);
```

6 Other Minor Utilities

In addition to the standard quality-control tools described in the previous sections, QoRTs also includes a number of other minor utilities intended to assist in data visualization, cleaning, and preparation for downstream analyses.

6.1 Optional: Generating a flattened annotation file

Before counting exons and splice junctions, QoRTs generates a set of non-overlapping exonic fragments out of all the exons in the genome annotation gtf file. It then assigns each exonic fragment a unique identifier. Similarly, it assigns every splice junction its own unique identifier. A gtf file listing all these genomic features and their unique identifiers can be created using the following command:

```
java -jar /path/to/jarfile/QoRTs.jar makeFlatGtf
                                                input.gtf
                                                flattened.gff
```

strandedness: You must use the `--stranded` option to create the flattened gff for use with stranded datasets. DO NOT mix stranded flattened gff with unstranded data, or vice versa.

DEXSeq: DEXSeq also requires a flattened annotation file, which is formatted similarly. In order to produce a flattened gff file that DEXSeq can read, add the `--DEXSeqFmt` option.

This gtf file conforms to the UCSC gff file definition, (found here: `http://genome.ucsc.edu/FAQ/FAQformat.html`). It will contain 4 different feature types (column 3): "aggregate_gene", "exonic_part", "splice_site", and "novel_splice_site". This file is an expanded variant of the flattened gff files produced by DEXSeq.

- *aggregate_gene* Aggregate genes are usually single genes. However, whenever genes overlap, the set of overlapping genes are combined into a single aggregate gene.
- *exonic_part*
- *splice_site*
- *novel_splice_site*

6.2 Generating genome browser tracks

In addition to the standard QC plots, which examine the data as a whole, it is sometimes desirable to be able to query and examine coverage information at specific genetic loci. In particular, when identifying candidate genes via genome-wide analyses, it is often vital to examine the locus for artifacts before carrying out costly and time-consuming validation experiments.

6.2.1 Generating wiggle tracks

QoRTs includes a utility to generate ".wig", or "wiggle plot" files. These wiggle plot files include counts for the mean coverage for each equal-sized window across the whole genome. These files are designed to be used with the UCSC browser or similar interfaces, and allow easy and intuitive visualization of your data.

```
java -jar /path/to/jarfile/QoRTs.jar bamToWiggle
                                                infile.bam
                                                trackName
                                                chromLengthFile
                                                outfilePrefix
```

The chromLengthFile is a simple tab-delimited text file that includes each chromosome in the first column and the chromosome's length (in base-pairs) in the second column. If the wiggle file is intended for use with a standard genome on the UCSC genome browser, then the UCSC utility `fetchChromSizes` should be used to generate this file. (see `http://genome.ucsc.edu/goldenPath/help/bigWig.html` for more information on `fetchChromSizes`, as well as information on how to compress your wig files into smaller and more efficient bigWig files)
Common options and flags for this function include:

- *–sizefactor 1.0*: A float value. All the coverage values will be divided by this factor. Useful for comparing two samples that may have different normalization factors.
- *–stranded*: Flag to indicate that data should be treated as stranded.
- *–stranded_fr_secondstrand*: Flag to indicate that the data is of the fr_secondstrand stranded library type. (See section 4 for more information on the two stranded library types)
- *–negativeReverseStrand*: If this flag is set, then the negative strand will be counted in negative numbers. This can be useful for plotting both strands in a single multiwig track, via a trackhub. (see `http://genome.ucsc.edu/goldenPath/help/trackDb/trackDbDoc.html`)

6.2.2 Merging wiggle tracks

QoRTs includes a utility for summing or averaging multiple wiggle files, either with or without normalization factors.

```
java -jar /path/to/jarfile/QoRTs.jar sumWiggles
                                          filelist.txt
                                          outfile.wig
```

The filelist.txt is a simple text file that contains the list of wiggle files to merge (one filename per line). *Optionally*, the filelist.txt can contain a second (tab-delimited) column which includes normalization factors for each wiggle file. If this is selected, the each wiggle file will be normalized by the factor given (ie, the coverage count will be divided by the corresponding factor, before being added to the total). Common options and flags for this function include:

- *–calcMean*: If this flag is raised, the utility will calculate the average rather than the total coverage for each window.
- *–ignoreSizeFactors*: By default the utility attempts to auto-detect the presence of normalization factors in the filelist.txt file. If this flag is raised, the utility will ignore all but the first column of filelist.txt, and set all size factors to 1.
- *–makeNegative*: If this flag is raised, the output will be multiplied by -1.

6.2.3 Generating splice-junction tracks

Splice junction counts can be made into a separate bed track using the command:

```
java -jar /path/to/jarfile/QoRTs.jar makeSpliceBed
                                          filelist.txt
                                          outfile.bed
```

Common options and flags for this function include:

- *–rgb*: The color to use for each bed entry.

6.2.4 Merging splice-junction tracks

ADD MORE INFO HERE: TODO: merging splice junction bed files.

6.3 Merging Count Data

For the purposes of quality control it is generally preferable to run QoRTs on each sample-bam individually, so that potential technical artifacts related to sequencing run or lane can be identified. However, for most purposes these "technical replicates" will be treated as a single sample by downstream analyses. Differential expression tools like *DESeq*, *DESeq2* [1], *DEXSeq* [3], and *EdgeR* [7] assume that each set of gene counts (or exon counts, for *DEXSeq*) is derived from a different biological sample.

Thus, the java utility includes a function for quickly and easily calculating merged sample-wise counts.

```
java -jar /path/to/jarfile/QoRTs.jar mergeAllCounts
                                     decoder.txt
                                     /path/to/qc/results/dir/
                                     ./merged/
```

Alternatively, the merger can be performed for a single sample directly, via the command:

```
java -jar /path/to/jarfile/QoRTs.jar mergeCounts
          ./CtrlDay1_RG1/,./CtrlDay1_RG2/,./CtrlDay1_RG3/
          ./merged/CD1/
```

The list of QC data directories must be separated by commas and contain no whitespace.

6.4 Importing data into other tools

In addition to providing quality control information, QoRTs also provides the requisite input files needed for the DESeq/DESeq2 [1], *DEXSeq* [3], and *EdgeR* [2,7,8] analysis tools. These files will be identical to those that would be generated by HTSeq (using the default "union rule" option).

All the data files can be found in the `qc.data.dir` directory. The files for use with *DESeq*, *DESeq2*, and *EdgeR* will be named `QC.geneCounts.formatted.for.DESeq.txt.gz` and the files for use with *DEXSeq* will be named `QC.exonCounts.formatted.for.DEXSeq.txt.gz`

7 References

[1] Simon Anders and Wolfgang Huber. Differential expression analysis for sequence count data. *Genome Biology*, 11:R106, 2010.

[2] Mark D. Robinson and Gordon K. Smyth. Moderated statistical tests for assessing differences in tag abundance. *Bioinformatics*, 23:2881, 2007.

[3] Simon Anders, Alejandro Reyes, and Wolfgang Huber. Detecting differential usage of exons from RNA-seq data. *Genome Research*, 22:2008, 2012.

[4] Alexander Dobin, Carrie A. Davis, Felix Schlesinger, Jorg Drenkow, Chris Zaleski, Sonali Jha, Philippe Batut, Mark Chaisson, and Thomas R. Gingeras. STAR: ultrafast universal RNA-seq aligner. *Bioinformatics*, 29(1):15–21, 2013.

[5] Thomas D. Wu and Serban Nacu. Fast and SNP-tolerant detection of complex variants and splicing in short reads. *Bioinformatics*, 26(7):873–881, 2010.

[6] Daehwan Kim, Geo Pertea, Cole Trapnell, Harold Pimentel, Ryan Kelley, and Steven Salzberg. Tophat2: accurate alignment of transcriptomes in the presence of insertions, deletions and gene fusions. *Genome Biology*, 14(4):R36, 2013.

[7] McCarthy DJ Robinson MD and Smyth GK. edger: a bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics*, 26:139–140, 2010.

[8] Davis J. McCarthy, Yunshun Chen, and Gordon K. Smyth. Differential expression analysis of multi-factor RNA-Seq experiments with respect to biological variation. *Nucleic Acids Research*, 40:4288–4297, 2012.

8 Session Information

The session information records the versions of all the packages used in the generation of the present document.

```
sessionInfo()

## R version 3.0.2 (2013-09-25)
## Platform: x86_64-unknown-linux-gnu (64-bit)
##
## locale:
## [1] C
##
## attached base packages:
## [1] parallel  stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] edgeR_3.4.2             limma_3.18.12           DESeq2_1.2.10
##  [4] RcppArmadillo_0.4.000.2 Rcpp_0.11.0             GenomicRanges_1.14.4
##  [7] XVector_0.2.0           IRanges_1.20.6          BiocGenerics_0.8.0
## [10] QoRTsExampleData_0.0.10 QoRTs_0.0.16            Cairo_1.5-5
## [13] knitr_1.6.1
##
## loaded via a namespace (and not attached):
##  [1] AnnotationDbi_1.24.0 Biobase_2.22.0       BiocStyle_1.0.0
##  [4] DBI_0.2-7            RColorBrewer_1.0-5   RSQLite_0.11.4
##  [7] XML_3.98-1.1         annotate_1.40.0      evaluate_0.5.5
## [10] formatR_0.10         genefilter_1.44.0    grid_3.0.2
## [13] highr_0.3            lattice_0.20-24      locfit_1.5-9.1
## [16] splines_3.0.2        stats4_3.0.2         stringr_0.6.2
## [19] survival_2.37-7      tools_3.0.2          xtable_1.7-1
```

9 Legal

NOTE: The Scala package includes (internally) the sam-JDK library (sam-1.113.jar), from picard tools. The MIT license and copyright information can be accessed using the command:

```
java -jar /path/to/jarfile/QoRTs.jar ? samjdkinfo
```