QoRTs Example Dataset Analysis Walkthrough

Stephen Hartley
National Human Genome Research Institute
National Institutes of Health

August 8 2014
Revised 2 March 2015
v0.2.11

Contents

1 Overview

The QoRTs software package is a fast, efficient, and portable multifunction toolkit designed to assist in the analysis, quality control, and data management of RNA-Seq datasets. Its primary function is to aid in the detection and identification of errors, biases, and artifacts produced by paired-end high-throughput RNA-Seq technology. In addition, it can produce count data designed for use with differential expression [1] and differential exon usage tools [2], as well as individual-sample and/or group-summary genome track files suitable for use with the UCSC genome browser (or any compatible browser).

This walkthrough intends provides step-by-step instructions on how to generate QC reports, generate advanced browser tracks, and run various differential regulation analyses using the QoRTs software package. This walkthrough uses a small example dataset to allow users to follow along on their own if desired, which can be found here: `https://dl.dropboxusercontent.com/u/103621176/qorts/exData/QoRTsFullExampleData.zip`.

The QoRTs package is composed of two parts: a java jar-file (for data processing) and a companion R package (for generating tables, figures, and plots). The 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. The entire QoRTs toolkit can be used in almost any operating system that supports java and R.

[1] Such as *DESeq*, *DESeq2* [1] or *edgeR* [2]
[2] Such as *DEXSeq* [3]

The most recent release of QoRTs is available on the QoRTs github page (http://github.com/hartleys/QoRTs). Additional help and documentation is available online (https://dl.dropboxusercontent.com/u/103621176/QoRTs/helpDocs/index.html). A comprehensive walkthrough covering the entire process from post-alignment all the way to differential expression analysis, along with a full example dataset and example output can be found online (https://dl.dropboxusercontent.com/u/103621176/qorts/exData/QoRTsFullExampleData.zip).

2 Using this Walkthrough

This walkthrough demonstrates how to use the QoRTs software package on one particular example dataset. However, many of the scripts and commands used here could be used on any RNA-Seq dataset with minimal modification. File locations will have to be modified, as well as the path to the java jar-file (in the example scripts, it is `QoRTsRelease/QoRTs.jar`).

Additionally, in this example walkthrough all commands are carried out in series. In actual use, it is generally recommended that separate QoRTs runs be executed in separate threads, or, if available, separate jobs run on a cluster job-queuing engine (such as SGE).

3 Requirements

Hardware: The java utility does the bulk of the data processing, and will generally require at least 4gb of RAM. In general at least 8gb is recommended, if available. 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 workstation.

Software: The QoRTs software package requires R version 3.0.2 or higher, as well as java 6 or higher.

Annotation: QoRTs requires transcript annotations in the form of a gtf file. If you are using a annotation guided aligner (which is STRONGLY recommended) 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[3] suitable for these purposes. However, any format that adheres to the gtf file specification[4] will work.

3.1 Recommendations

Clipping: For the purposes of Quality Control, it is generally best if reads are NOT hard-clipped prior to alignment. This is because hard clipping, espeically variable hard-clipping from both the start and end of reads, makes it impossible to determine sequencer cycle from the aligned bam files, which in turn can obfuscate cycle specific artifacts, biases, errors, and effects. If undesired sequence must be removed,

[3] Which can be acquired from the Ensembl website at http://www.ensembl.org
[4] See the gtf file specification at http://genome.ucsc.edu/FAQ/FAQformat.html

it is generally preferred to replace such nucleotides with N's, as this preserves cycle information. Note that many advanced RNA-Seq aligners will "soft clip" nonmatching sequence that occurs on the read ends, so hard-clipping low quality sequence is generally unnessessary and may reduce mapping rate and accuracy.

Replicates: 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. In general, it is recommended that samples for a particular study be multiplexed and merged into "balanced" combined libraries, each containing equal numbers of each biological condition. If necessary, these combined libraries can be run across multiple sequencer lanes or runs to achieve the desired read depth on each sample.

In the example dataset, all samples were merged into a single combined library and run on three sequencer lanes. The demultiplexed results were aligned seperately and produce seperate bam files. For example, sample "SAMP1" is composed of "SAMP1_RG1.bam", "SAMP1_RG2.bam", and "SAMP1_RG3.bam". If preferred, these "technical replicates" can be merged prior to alignment and analyzed seperately (see section 6 for details on the `--readGroup` option).

4 Preparations

There are a number of processing steps that must occur prior to the creation of usable bam files. The specifics of such steps is beyond the scope of this walkthrough. We will briefly go over the required steps here:

4.1 Alignment

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.

4.2 Sorting

For paired-end data, QoRTs requires that the bam files be sorted by either read name or by position. Sorting can be accomplished via the samtools or novosort tools (which are NOT included with QoRTs). Sorting is unnecessary for single-end data.

To sort by position:

```
samtools sort SAMP1_RG1.raw.bam SAMP1_RG1
```

OR

```
novosort SAMP1_RG1.raw.bam > SAMP1_RG1.bam
```

The example dataset is already sorted by position.

4.3 Bam file decoder

Several QoRTs functions will require a "decoder" file, which describes each sample and all of its technical replicates (if any). All of the columns are optional except for unique.ID.

Fields:

- *unique.ID*: A unique identifier for the row. THIS IS THE ONLY MANDATORY FIELD.
- *lane.ID*: The ID of the lane or batch. By default this will be set to "UNKNOWN".
- *group.ID*: The ID of the "group". For example: "Case" or "Control". By default this will be set to "UNKNOWN".
- *sample.ID*: The ID of the biological sample from which the data originated. Each sample can have multiple rows, representing technical replicates (in which the same sample is sequenced on multiple lanes or runs). By default QoRTs will assume that every row comes from a separate sample, and will thus set the sample.ID to equal the unique.ID.
- *qc.data.dir*: The directory in which the java utility is to save all the QC data. If this column does not exist, by default it will be set to the unique.ID.
- *input.read.pair.count*: The number of reads in the original fastq file, prior to alignment.
- *multi.mapped.read.pair.count*: The number of reads that were multi-mapped by the aligner.

In addition, the decoder can contain any other additional columns as desired, as long as all of the column names are distinct.

5 Example dataset

The example dataset is derived from a set of rat pineal gland samples, which were multiplexed and sequenced across six sequencer lanes. All samples are paired-end, 2x101 base-pair, strand-specific RNA-Seq. They were ribosome-depleted using the "Ribo-zero Gold" protocol and aligned via RNA-STAR.

For the sake of simplicity, the example dataset was limited to only six samples and three lanes. However, the bam files alone would still occupy 18 gigabytes of disk space, which would make it unsuitable for distribution as an example dataset. To further reduce the example bamfile sizes, only reads that mapped to chromosomes chr14, chr15, chrX, and chrM were included. Additionally, all the selected chromosomes EXCEPT for chromosome 14 were randomly downsampled to 30 percent of their original read counts.

THIS DATASET IS INTENDED FOR DEMONSTRATION AND TESTING PURPOSES ONLY. Due to the various alterations that have been made to reduce file sizes and improve portability, it is really not suitable for any actual analyses.

5.1 Overview of example files

The "input" data files are all found in the `inputData/` directory:

- `inputData/annoFiles/anno.gtf.gz`: A truncated gtf file generated by extracting chromosomes 14, 15, X and M from the ensembl `Rattus_norvegicus.RGSC3.4.69.gtf` transcript annotation file.
- `inputData/annoFiles/chrom.sizes`: A truncated chromosome sizes file generated by extracting chromosomes 14, 15, X and M from the chrom.sizes file generated by the UCSC `fetchChromSizes` utility.
- `inputData/annoFiles/decoder.bySample`: A sample decoder, which defines each sample's biological condition (CASE or CTRL).
- `inputData/annoFiles/decoder.byUID`: A decoder that matches sample.ID to unique.ID, matching technical replicates to their sample of origin. It also includes (optional) biological condition information as well as input read pair counts and multimapping read pair counts acquired from the aligner output.
- `inputData/annoFiles/sampleID.list.txt`: A simple text file containing the list of samples.
- `inputData/annoFiles/uniqueID.list.txt`: A simple text file containing the list of technical replicates.

In addition, bam files for the 3 technical replicates for each of the 6 example samples are included in the `inputData/bamFiles/` directory.

5.2 Following along on your own

In order to follow along with this walkthrough, download the example dataset from (https://dl.dropboxusercontent.com/u/103621176/qorts/exData/QoRTsFullExampleData.zip).

This walkthrough assumes that analysis is being done in a Linux-like environment (bash). Most commands can be adapted for other enviroments relatively easily, as QoRTs is not platform dependent.

All commands are executed from the root directory of the example dataset: `QoRTsFullExampleData/`.

This example walkthrough will additionally require the installation of R packages: *MASS*, *Cairo*, *png*, and bioconductor packages: *DESeq2*, *edgeR*, and *DEXSeq*.

These packages can be installed with the R commands:

```
install.packages("Cairo")
install.packages("MASS")
install.packages("png")

source("http://bioconductor.org/biocLite.R")
biocLite()
biocLite("DESeq2")
biocLite("DEXSeq")
biocLite("edgeR")
```

QoRTs R Package: It is generally preferable to download the most recent release of QoRTs [5], however the version of QoRTs used to generate the example output data is included with the example dataset. This version of QoRTs can be installed using the console command:

```
R CMD INSTALL QoRTsRelease/QoRTs_0.2.11.tar.gz
```

Again, this command must be executed from the root directory of the example dataset (`QoRTsFullExampleData/`).

The example dataset comes with all output already generated. In order to delete this pre-generated output and "start from scratch", run the script:

```
sh exampleScripts/step0/step0.reset.sh
```

[5] Found here: https://github.com/hartleys/QoRTs/releases

6 Initial Data Processing (STEP 1)

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 minutes to process 1 million read-pairs.

To perform this analysis on `SAMP1_RG1.bam`, use the command:

```
mkdir outputData/qortsQcData/SAMP1_RG1/

java -Xmx1G -jar QoRTsRelease/QoRTs.jar QC \
                    --stranded --coordSorted \
                    inputData/bamFiles/SAMP1_RG1.bam \
                    inputData/annoFiles/anno.gtf.gz \
                    outputData/qortsQcData/SAMP1_RG1/
```

To run the example set on each bam file, you can use the commands:

```
while read line
do
  mkdir outputData/qortsData/$line/
  java -Xmx1G -jar QoRTsRelease/QoRTs.jar \
                QC \
                --stranded \
                --coordSorted \
                inputData/bamFiles/$line.bam \
                inputData/annoFiles/anno.gtf.gz \
                outputData/qortsData/$line/
done < "inputData/annoFiles/uniqueID.list.txt"
```

Or you can run the provided example script:

```
sh exampleScripts/step1/step1.all.qortsRun.sh
```

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

```
java -jar QoRTsRelease/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` option must be used.

- *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.
- *Read Groups:* Depending on the production pipeline, each biological sample may be run across multiple sequencer lanes. These seperate files can be merged together either before or after analysis with QoRTs (and maybe even before alignment). However, if the merger occurs before analysis with QoRTs, then each bam file will consist of multiple seperate lanes or runs. In this case, it is STRONGLY recommended that seperate QC runs be performed on each "read group", using the `--readGroup` option. This will prevent run- or lane-specific biases, artifacts, or errors from being obfuscated.
- *Read Sorting:* For paired-end data reads must be sorted. By default, QoRTs assumes that the bam file is sorted by *NAME*. If the bam file is instead sorted by position, then the `--coordSorted` option must be used. Sorting is unnecessary for single-end data.
- *Single-end vs paired-end:* By default, QoRTs assumes the input bam file consists of paired-end data. For single-end data, the `--isSingleEnd` option must be used.
- *'Wiggle' Browser Tracks:* '.wig' browser tracks can also be produced in this step. See Section 8 for more information.
- *Junction Browser Tracks:* '.bed' junction depth browser tracks can also be produced in this step. See Section 10 for more information.

6.1 Memory Usage

Memory usage: The QoRTs QC utility requires at least 4gb or RAM for most genomes / datasets. Larger genomes, genomes with more annotated genes/transcripts, or larger bam files may require more RAM. You can set the maximum amount of RAM allocated to the JVM using the options `-Xmx4000M`. This should be included before the `-jar` in the command line. This option can be used with any and all of the QoRTs java utilities. Because the example dataset and genome is so small, 1 gigabyte of RAM is more than sufficient.

7 Generate QC plots (Step 2)

Once the data has been processed, you can read the QC data produced by QoRTs into R using the command:

```
library(QoRTs);

#Read in the QC data:
res <- read.qc.results.data("outputData/qortsData/",
            decoder.files = "inputData/annoFiles/decoder.byUID.txt",
            calc.DESeq2 = TRUE, calc.edgeR = TRUE);
```

Once you have read in the QC data, you can build all sorts of plots. See Figure 1 for one example[6].

EXAMPLE 1: The `makeMultiPlot.all` can be used to automatically generate a full battery of multi-plot figures:

```
makeMultiPlot.all(res,
                  outfile.dir = "outputData/qortsPlots/summaryPlots/",
                  plot.device.name = "png");
```

EXAMPLE 2: Some users may find the large png files difficult to read. QoRTs offers multi-page pdf reports as an alternative, simply by using the `plot.device.name` parameter:

```
makeMultiPlot.all(res,
                   outfile.dir = "outputData/qortsPlots/summaryPDFs/",
                   plot.device.name = "pdf");
```

EXAMPLE 3: To print all the basic plots as seperate pngs, use the command:

```
makeMultiPlot.basic(res,
                   outfile.dir = "outputData/qortsPlots/basicPlots/",
                   separatePlots = TRUE);
```

Note that this produces a huge number of png files.

A full description of all quality control plots produced by these commands is beyond the scope of this walkthrough. See the QoRTs package vignette for more information.

7.1 Extract Size Factors

If DESeq2 and/or edgeR are installed, QoRTs will automatically generate "library size factors" using the algorithms provided by each. These "size factors" are used to "normalize" all samples to a common, comparable scale. This is necessary in order to directly compare between samples, and thus size factors are required for several downstream analyses. To extract these size factors, use the R commands:

[6] Note: to reduce the pdf file size, the resolution is lower than the default value.

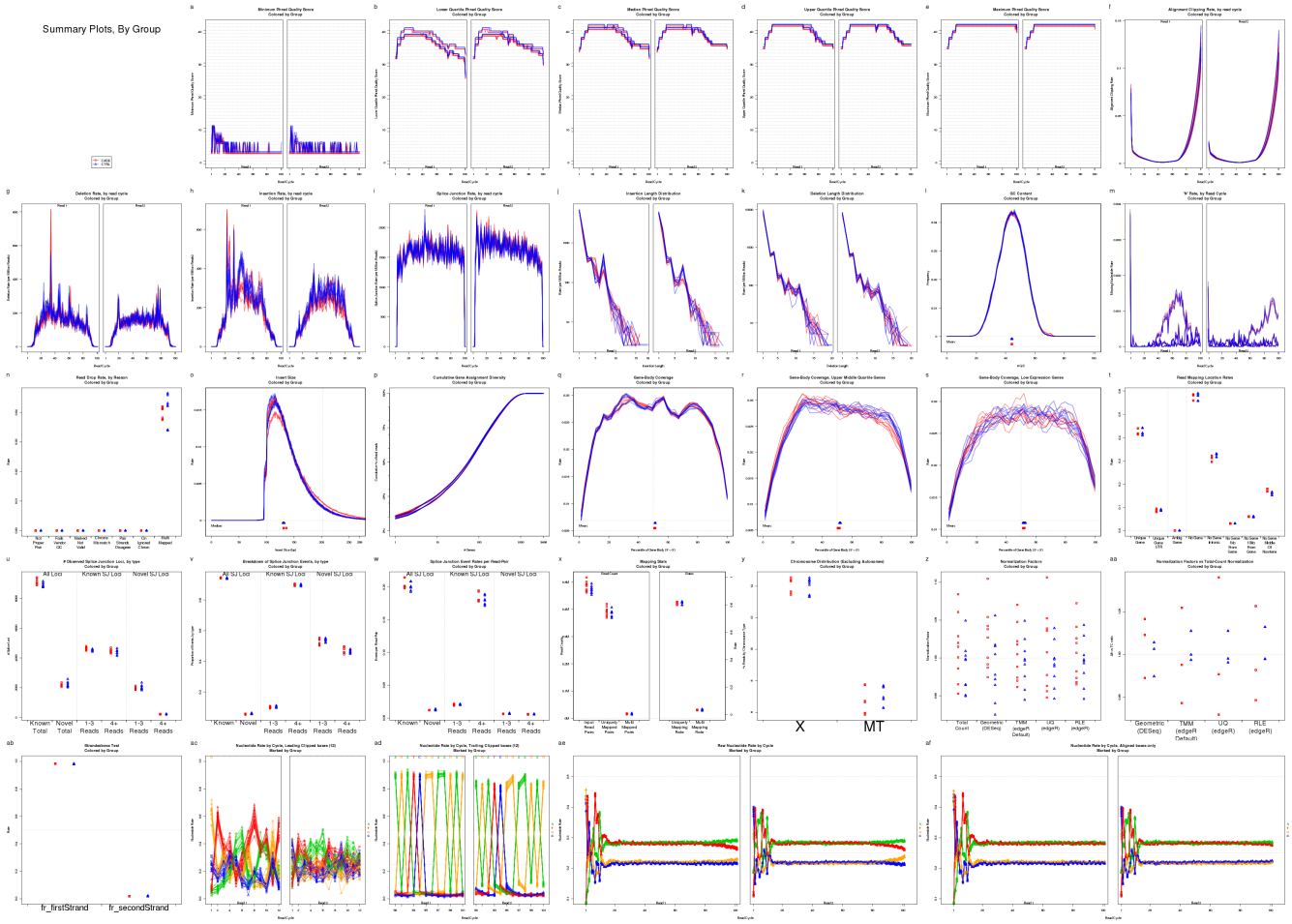

Figure 1: An example combined multi-plot. In this example, replicates are differentiated by color and pch symbol based on the group.ID (ie CASE or CTRL).

```
sizeFactors <- res@calc.data[["norm.factors.bySample"]];
sizeFactors.GEO <- data.frame(sample.ID = sizeFactors$sample.ID,
                              size.factor = sizeFactors$Norm_Geo);
write.table(sizeFactors.GEO,file = "outputData/sizeFactors.GEO.txt",
            row.names=F,col.names=TRUE,sep="\t",quote=F);
```

8 Generate Wiggle Tracks (STEP 3)

To generate "wiggle" tracks for a replicate, you can use the command:

```
  java -Xmx1G -jar QoRTsRelease/QoRTs.jar \
               bamToWiggle \
               --stranded \
               --windowSize 100 \
               --negativeReverseStrand \
               --includeTrackDefLine \
               --rgbColor 0,0,0 \
               --coordSorted \
               inputData/bamFiles/SAMP1_RG1.bam \
               SAMP1_RG1 \
               inputData/annoFiles/chrom.sizes \
               outputData/qortsData/SAMP1_RG1/QC.wiggle
```

To generate wiggle tracks for all bam files:

```
while read line
do
  java -Xmx1G -jar QoRTsRelease/QoRTs.jar \
               bamToWiggle \
               --stranded \
               --windowSize 100 \
               --negativeReverseStrand \
               --includeTrackDefLine \
               --rgbColor 0,0,0 \
               --coordSorted \
               inputData/bamFiles/$line.bam \
               $line \
               inputData/annoFiles/chrom.sizes \
               outputData/qortsData/$line/QC.wiggle
done < "inputData/annoFiles/uniqueID.list.txt"
```

These tracks can be loaded and viewed in the UCSC genome browser. See Figure 2 for an example.

Or run the provided example script:

```
sh exampleScripts/step3/step3.all.qortsWig.sh
```

More details on parameters and options available for this function can be accessed with the commands:

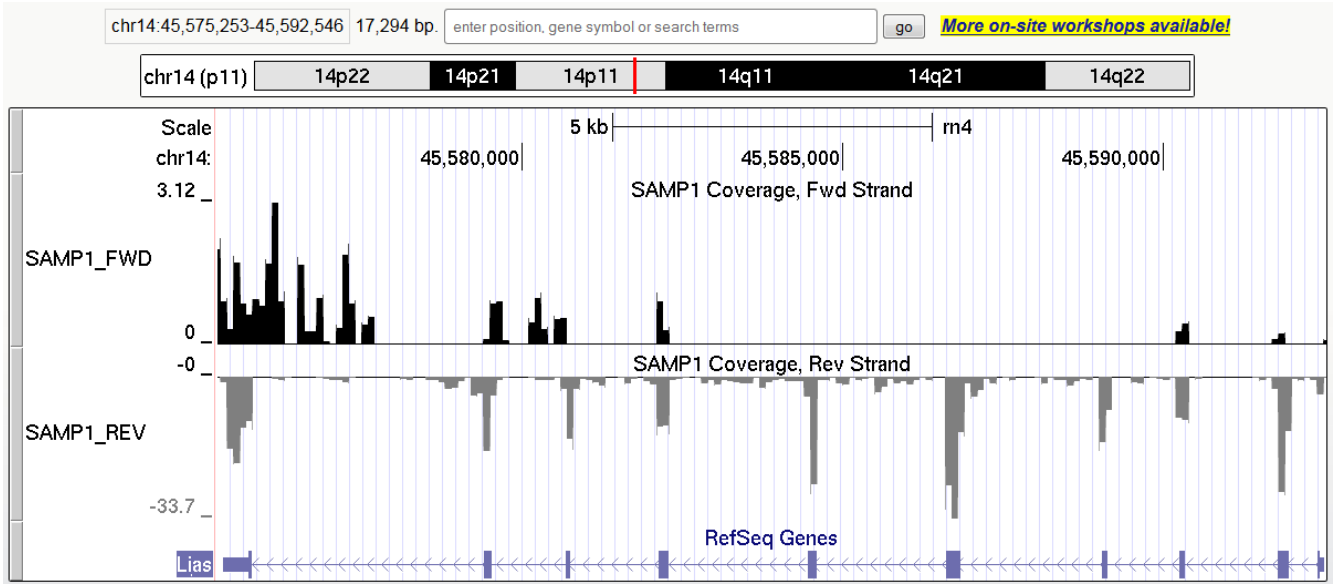

Figure 2: A pair of wiggle tracks generated for SAMP1. The top wiggle track plots the average coverage depth over the forward strand, the lower wiggle track plots the average coverage depth over the reverse strand.

```
java -jar  QoRTsRelease/QoRTs.jar bamToWiggle --man
```

Wiggle tracks can also be generated during the initial processing step (See Section 6). However, this requires the `--chromSizes` parameter be set to the chrom.sizes file. The additional option `trackTitlePrefix` can be used to set the wiggle track's title.

So, for example, to generate all QC data and wiggle tracks simultaniously in a single bam-file pass:

```
java -Xmx1G -jar QoRTsRelease/QoRTs.jar QC \
                    --stranded --coordSorted \
                    --chromSizes inputData/annoFiles/chrom.sizes \
                    --trackTitlePrefix SAMP1_RG1_WIGGLE  \
                    inputData/bamFiles/SAMP1_RG1.bam \
                    inputData/annoFiles/anno.gtf.gz \
                    outputData/qortsQcData/SAMP1_RG1/
```

9 Merge Technical Replicates (STEP 4)

In many RNA-Seq datasets, individual samples/libraries are run across multiple sequencing lanes, producing "technical replicates" of the same original biological samples. For the purposes of quality control it is generally best to run QC tools separately on each technical replicate, so that lane- or run-specific errors and artifacts are not obscured. However, for the purposes of most downstream analyses, it is generally preferable to merge the technical replicates for each sample. Thus, QoRTs contains methods allowing count data to be combined across technical replicates.

If the dataset contains no technical replicates then this step is unnecessary.

To merge the count data of all technical replicates:

```
java -jar QoRTsRelease/QoRTs.jar \
                mergeAllCounts \
                outputData/qortsData/ \
                inputData/annoFiles/decoder.byUID.txt \
                outputData/countTables/
```

This requires a decoder file, identical to the one used for QC in section 7. The only required fields are unique.ID and sample.ID.

Alternatively, an individual set of technical replicates can be merged using the command:

```
java -jar QoRTsRelease/QoRTs.jar \
                mergeCounts \
                outputData/qortsData/SAMP1_RG1/,outputData/qortsData/SAMP1_RG2/,o\
utputData/qortsData/SAMP1_RG3/ \
                outputData/countTables/SAMP1/
```

More details on parameters and options available for this function can be accessed with the commands:

```
java -jar  QoRTsRelease/QoRTs.jar mergeCounts --man
```

and

```
java -jar  QoRTsRelease/QoRTs.jar mergeAllCounts --man
```

10 Generate Junction Tracks (STEP 5)

For the purposes of examining transcript abundances as well as visually confirming the results of transcript deconvolution or transcript assembly tools, QoRTs provides functions for generating genome browser tracks that display splice junction coverage counts. These can be generated for an individual replicate during the initial processing step (See Section 6), by adding the `--addFunctions makeJunctionBed` option.

See Figure 3 for an example, shown alongide wiggle tracks generated in Section 9.

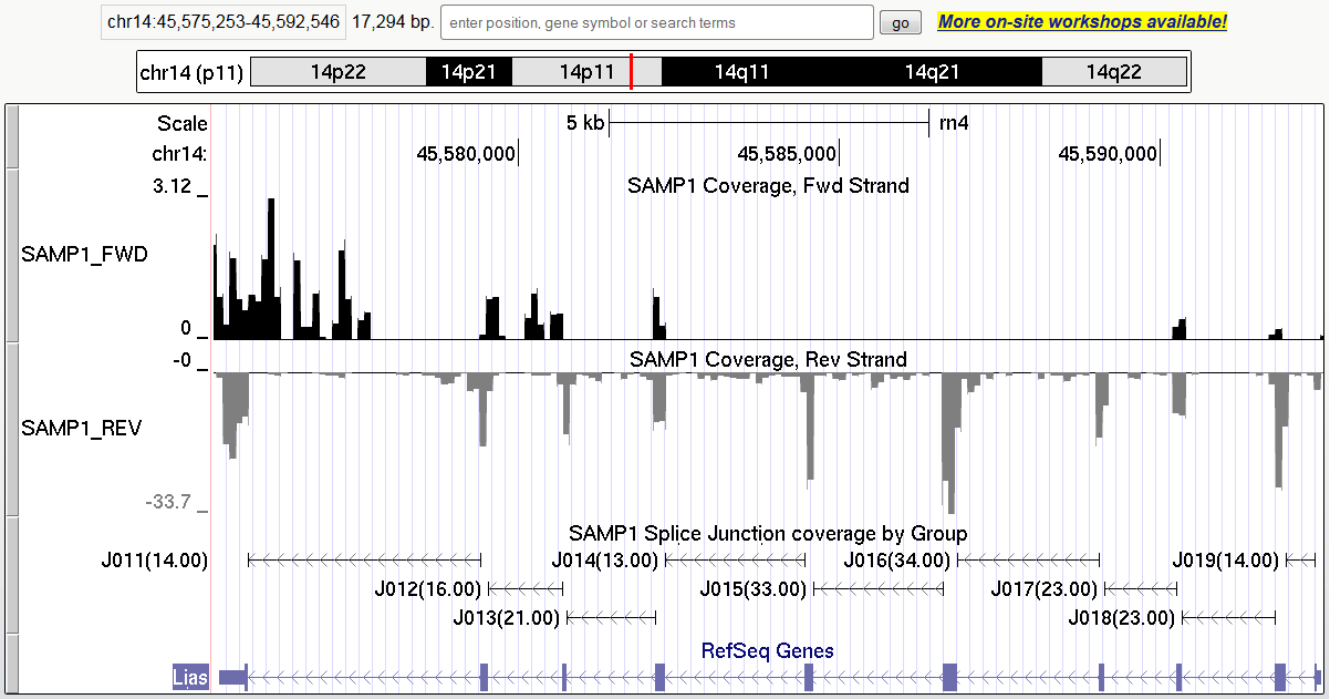

Figure 3: An example of the tracks that can be produced in this section. Each annotated splice junction locus has a identifier unique for each gene (example: J017). After the identifier, the number of reads (or read-pairs) that bridge the given splice junction locus is listed in parentheses

10.1 Generate "flat" annotation files

DEXSeq requires a "flattened" gff files, in which overlapping genes are compiled and a flat set of exons is generated and assigned unique identifiers. QoRTs has extended this notation to additionally assign unique identifiers to every splice junction as well. These assignments can be generated and accessed using the command:

```
java -Xmx1G -jar  QoRTsRelease/QoRTs.jar \
                  makeFlatGff \
```

```
                    --stranded \
                    inputData/annoFiles/anno.gtf.gz \
                    outputData/forJunctionSeq.gff.gz
```

This gff file can be used to create splice junction tracks (but is NOT required to do so).

To generate a flat annotation file that is formatted for use with the *DEXSeq* package, use the command:

```
java -Xmx1G -jar  QoRTsRelease/QoRTs.jar \
                  makeFlatGff \
                  --stranded \
                  --DEXSeqFmt \
                  inputData/annoFiles/anno.gtf.gz \
                  outputData/forDEXSeq.gff.gz
```

More details on parameters and options available for this function can be accessed with the command:

```
java -jar  QoRTsRelease/QoRTs.jar makeFlatGff --man
```

10.2 Make Sample Junction Tracks

QoRTs includes utilitys for creating splice junction coverage tracks for individual samples.

A splice junction coverage count bed file can be generated for sample SAMP1 using the command:

```
  java -Xmx1G -jar QoRTsRelease/QoRTs.jar \
       makeJunctionTrack \
       --nonflatgtf \
       --stranded \
       --filenames outputData/countTables/SAMP1/QC.spliceJunctionAndExonCounts.\
forJunctionSeq.txt.gz \
       inputData/annoFiles/anno.gtf.gz \
       outputData/countTables/SAMP1/QC.junctionBed.known.bed.gz
```

Or, to generate this file for every sample:

```
while read line
do
  java -Xmx1G -jar QoRTsRelease/QoRTs.jar \
       makeJunctionTrack \
       --nonflatgtf \
       --stranded \
       --filenames outputData/countTables/$line/QC.spliceJunctionAndExonCounts.\
forJunctionSeq.txt.gz \
       inputData/annoFiles/anno.gtf.gz \
       outputData/countTables/$line/QC.junctionBed.known.bed.gz
done < "inputData/annoFiles/sampleID.list.txt"
```

More details on parameters and options available for this function can be accessed with the command:

```
java -jar  QoRTsRelease/QoRTs.jar makeJunctionTrack --man
```

10.3 Filter and Merge Novel Junctions

The splice junction count files in Section 10.2 only include annotated splice junctions. It may be desirable to visually inspect novel splice junctions as well. However, some aligners may produce large numbers of extremely-low-coverage splice junctions. To filter out such junctions and produce unique identifiers for every splice junction locus that passes the filter, use the command:

```
java -Xmx1G -jar QoRTsRelease/QoRTs.jar \
                 mergeNovelSplices  \
                 --minCount 100 \
                 --stranded \
                 outputData/countTables/ \
                 outputData/sizeFactors.GEO.txt \
                 inputData/annoFiles/anno.gtf.gz \
                 outputData/countTables/
```

This will perform two functions: first it will produce a "flattened" gff file containing unique identifiers for all splice junctions that pass the given filter. Secondly, it produces new splice junction count files for each sample, which match the new flattened gtf.

More details on parameters and options available for this function can be accessed with the command:

```
java -jar  QoRTsRelease/QoRTs.jar mergeNovelSplices --man
```

10.4 Make sample junction tracks with novel junctions

Once the flattened annotation and count files which include the novel splice junctions has been generated, a bed file can be produces which includes all such novel splices. Note that identifier codes of novel splice junctions will be named with N's.

```
while read line
do
  java -Xmx1G -jar QoRTsRelease/QoRTs.jar \
       makeJunctionTrack \
       --filenames outputData/countTables/$line/QC.spliceJunctionAndExonCounts.withNo\
vel.forJunctionSeq.txt.gz \
       outputData/countTables/withNovel.forJunctionSeq.gff.gz \
       outputData/countTables/$line/QC.junctionBed.withNovel.bed.gz
done < "inputData/annoFiles/sampleID.list.txt"
```

More details on parameters and options available for this function can be accessed with the command:

```
java -jar  QoRTsRelease/QoRTs.jar makeJunctionTrack --man
```

11 Generate Summary Genome Tracks (STEP 6)

See Figure 4 for an example.

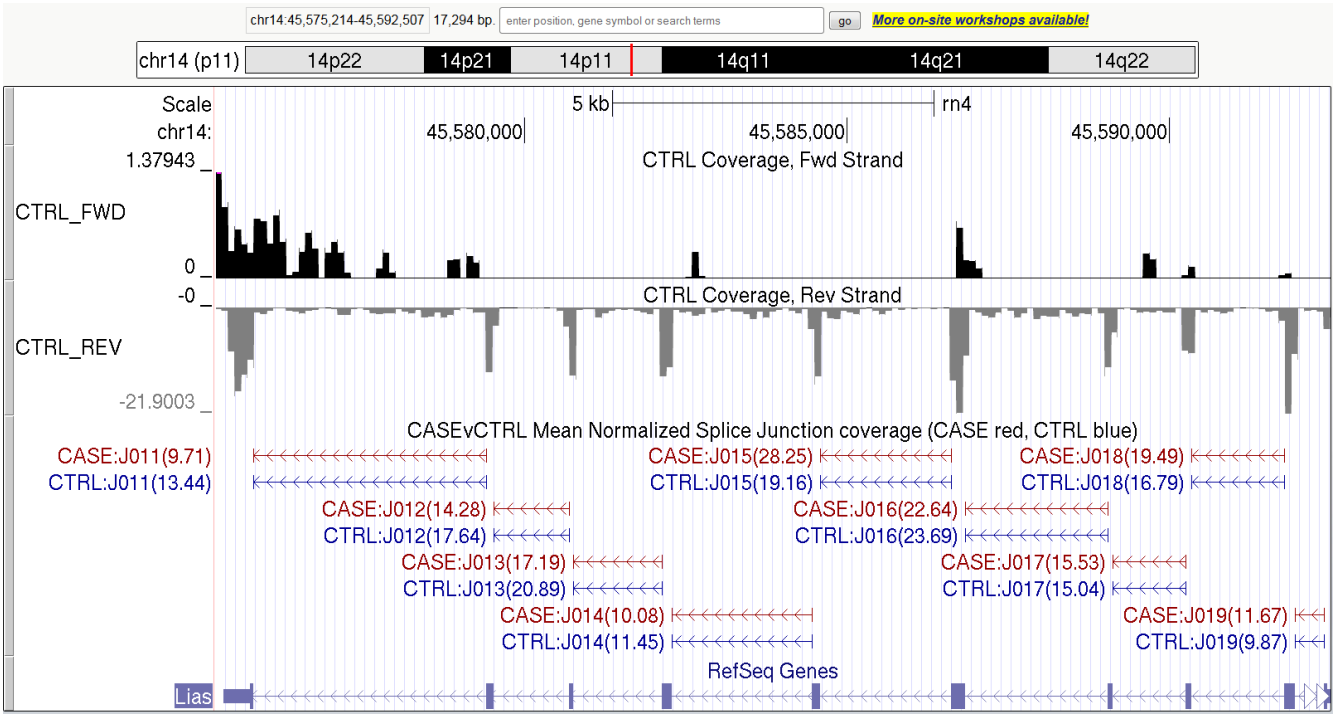

Figure 4: An example of the tracks that can be produced in Section 11. The first track ("CTRL_FWD") shows the mean normalized coverage depth for 100-base-pair windows for the forward (genomic) strand. The second track reveals the coverage depth on the negative strand (plotted as negative values). The third track ("CASEvCTRL Mean Normalized Splice Junction coverage") displays the locus ID for each splice junction, along with the mean normalized coverage bridging each splice junction locus in cases and in controls. Note that cases and controls are overlaid and colored distinctly in the junction track.

11.1 Make summary wiggle tracks

QoRTs can generate compiled summary wiggle tracks, which display the mean normalized read-pair coverage across multiple samples, over equal-sized windows across the entire genome.

Because there are two sets of wiggle files: forward and reverse strand (because this is stranded data), this requires two separate commands:

```
java -Xmx1G -jar QoRTsRelease/QoRTs.jar \
      mergeWig \
      --calcMean \
      --infilePrefix outputData/countTables/ \
      --infileSuffix /QC.wiggle.fwd.wig.gz \
      --sampleList SAMP1,SAMP2,SAMP3 \
      --sizeFactorFile outputData/sizeFactors.GEO.txt \
      outputData/mergedTracks/CASE.fwd.wig.gz

java -Xmx1G -jar QoRTsRelease/QoRTs.jar \
      mergeWig \
      --calcMean \
      --infilePrefix outputData/countTables/ \
      --infileSuffix /QC.wiggle.rev.wig.gz \
      --sampleList SAMP1,SAMP2,SAMP3 \
      --sizeFactorFile outputData/sizeFactors.GEO.txt \
      outputData/mergedTracks/CASE.rev.wig.gz
```

And, to do the same thing to the control set:

```
java -Xmx1G -jar QoRTsRelease/QoRTs.jar \
      mergeWig \
      --calcMean \
      --infilePrefix outputData/countTables/ \
      --infileSuffix /QC.wiggle.fwd.wig.gz \
      --sampleList SAMP4,SAMP5,SAMP6 \
      --sizeFactorFile outputData/sizeFactors.GEO.txt \
      outputData/mergedTracks/CTRL.fwd.wig.gz

java -Xmx1G -jar QoRTsRelease/QoRTs.jar \
      mergeWig \
      --calcMean \
      --infilePrefix outputData/countTables/ \
      --infileSuffix /QC.wiggle.rev.wig.gz \
      --sampleList SAMP4,SAMP5,SAMP6 \
      --sizeFactorFile outputData/sizeFactors.GEO.txt \
      outputData/mergedTracks/CTRL.rev.wig.gz
```

This tool is very flexible, and can be run on a number of different parameterizations. The example script `exampleScripts/step6/step6A.makeMergedWiggles.sh` contains a number of example variations.

More details on parameters and options available for this function can be accessed with the command:

```
java -jar  QoRTsRelease/QoRTs.jar mergeWig --man
```

11.2 Make summary junction tracks

QoRTs can also generate compiled summary junction-covarage tracks, which display the mean normalized read-pair coverage across multiple samples for all annotated splice junctions.

This can be achieved with the command:

```
java -jar QoRTsRelease/QoRTs.jar \
      makeJunctionTrack \
      --calcMean \
      --stranded \
      --infilePrefix outputData/countTables/ \
      --infileSuffix /QC.spliceJunctionAndExonCounts.forJunctionSeq.txt.gz \
      --sampleList SAMP1,SAMP2,SAMP3 \
      --sizeFactorFile outputData/sizeFactors.GEO.txt \
      outputData/forJunctionSeq.gff.gz \
      outputData/mergedTracks/CASE.bed.gz
```

To do the same thing for the CTRL group:

```
java -jar QoRTsRelease/QoRTs.jar \
      makeJunctionTrack \
      --calcMean \
      --stranded \
      --infilePrefix outputData/countTables/ \
      --infileSuffix /QC.spliceJunctionAndExonCounts.forJunctionSeq.txt.gz \
      --sampleList SAMP4,SAMP5,SAMP6 \
      --sizeFactorFile outputData/sizeFactors.GEO.txt \
      outputData/forJunctionSeq.gff.gz \
      outputData/mergedTracks/CTRL.bed.gz
```

A similar command can be used to include novel splice junctions, just like in section 10.4:

```
java -Xmx1G -jar QoRTsRelease/QoRTs.jar \
      makeJunctionTrack \
      --calcMean \
      --stranded \
      --infilePrefix outputData/countTables/ \
      --infileSuffix /QC.spliceJunctionAndExonCounts.withNovel.forJunctionSeq.txt.gz \
      --sampleList SAMP4,SAMP5,SAMP6 \
      --sizeFactorFile outputData/sizeFactors.GEO.txt \
      outputData/countTables/withNovel.forJunctionSeq.gff.gz \
      outputData/mergedTracks/CTRL.withNovel.bed.gz
```

This tool is particularly flexible, and can be run on a number of different parameterizations. The example script `exampleScripts/step6/step6A.makeMergedWiggles.sh` contains a number of example variations.

More details on parameters and options available for this function can be accessed with the command:

```
java -jar  QoRTsRelease/QoRTs.jar makeJunctionTrack --man
```

11.3 Combine summary junction tracks

Combined junction tracks can be generated displaying the seperate groups in different colors. First you must use the --rgb parameter to tell QoRTs to assign the given color to each group:

```
#Set cases to "red":
java -Xmx1G -jar QoRTsRelease/QoRTs.jar \
     makeJunctionTrack \
     --calcMean \
     --stranded \
     --rgb 150,0,0 \
     --title CASE \
     --infilePrefix outputData/countTables/ \
     --infileSuffix /QC.spliceJunctionAndExonCounts.withNovel.forJunctionSeq.txt.gz \
     --sampleList SAMP1,SAMP2,SAMP3 \
     --sizeFactorFile outputData/sizeFactors.GEO.txt \
     outputData/countTables/withNovel.forJunctionSeq.gff.gz \
     outputData/mergedTracks/CASE.withNovel.red.bed.gz

#And then set CONTROLS to "blue":
java -Xmx1G -jar QoRTsRelease/QoRTs.jar \
     makeJunctionTrack \
     --calcMean \
     --stranded \
     --rgb 0,0,150 \
     --title CTRL \
     --infilePrefix outputData/countTables/ \
     --infileSuffix /QC.spliceJunctionAndExonCounts.withNovel.forJunctionSeq.txt.gz \
     --sampleList SAMP4,SAMP5,SAMP6 \
     --sizeFactorFile outputData/sizeFactors.GEO.txt \
     outputData/countTables/withNovel.forJunctionSeq.gff.gz \
     outputData/mergedTracks/CTRL.withNovel.blue.bed.gz
```

These files can be merged simply by concatenation followed by sorting by the first and second columns. In linux this can be accomplished with the commands:

```
zcat outputData/mergedTracks/CASE.withNovel.red.bed.gz \
     outputData/mergedTracks/CTRL.withNovel.blue.bed.gz | \
     sort -k1,1 -k2,2n | gzip - > outputData/mergedTracks/CASE.vs.CTRL.bed.gz
```

11.4 Advanced Users

See Figure 5 for an example of how these summary tracks can be viewed via TrackHubs. The use of TrackHubs is beyond the scope of this manual. See the UCSC track database definition for more information (http://genome.ucsc.edu/goldenpath/help/trackDb/trackDbDoc.html).

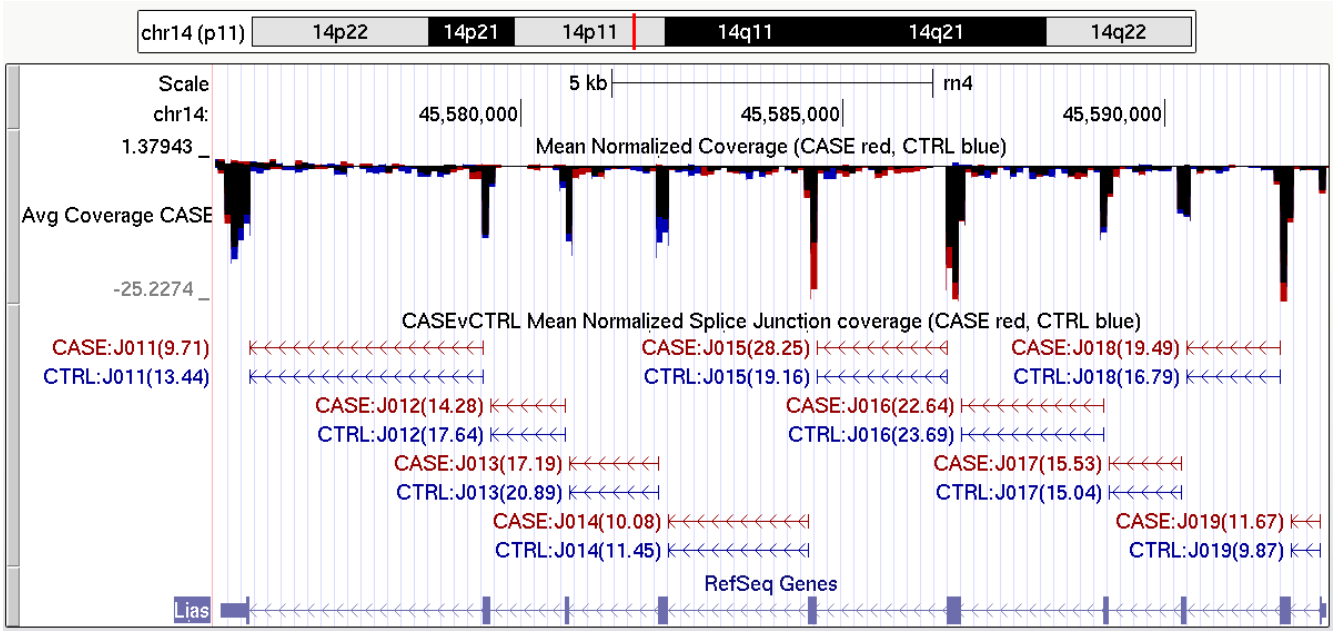

Figure 5: An example demonstrating how summary tracks can be viewed via the use of UCSC browser trackhubs. The aggregate "multiWig" track displays both cases and controls overlaid on top of one another in different colors. In addition, the forward and reverse strands are also overlaid, with the reverse strand displayed below zero.

A browser session that includes the trackhub shown above can be found at:

http://genome.ucsc.edu/cgi-bin/hgTracks?hgS_doOtherUser=submit&hgS_otherUserName=stephen.hartley&hgS_otherUserSessionName=rn4_QoRTs_Example

The hub itself can be found here:

https://dl.dropboxusercontent.com/u/103621176/QoRTs/exTrackHub/hub.txt

12 Perform differential expression analyses (STEP 7)

QoRTs produces output that is designed to be read by several different analyses tools designed to detect differential gene expression or other forms of differential regulation. It is designed to produce data identical to the data created by the HTSeq package.

12.1 Perform analysis with DESeq2

DESeq2 [1] is designed to detect gene-level differential expression in RNA-Seq data. As part of its initial processing QC run (see section 6), QoRTs produces count files similar to those expected by *DESeq2*.

The merged output produced by QoRTs mimics the output of HTSeq. Thus, we use the same syntax found in section 1.2.4 of the *DESeq2* vignette (the section titled "HTSeq Input").

```
suppressPackageStartupMessages(library(DESeq2));

decoder.bySample <- read.table(
                    "inputData/annoFiles/decoder.bySample.txt",
                    header=T,stringsAsFactors=F);

directory <- "outputData/countTables/";
sampleFiles <- paste0(
                    decoder.bySample$sample.ID,
                    "/QC.geneCounts.formatted.for.DESeq.txt.gz"
                  );
sampleCondition <- decoder.bySample$group.ID;
sampleName <- decoder.bySample$sample.ID;

sampleTable <- data.frame(sampleName = sampleName,
                          fileName = sampleFiles,
                          condition = sampleCondition);

dds <-  DESeqDataSetFromHTSeqCount(sampleTable = sampleTable,
                                   directory = directory,
                                   design = ~ condition);

dds;
```

Once the data is loaded into a `DESeqDataSet`, all desired *DESeq2* analyses can then be run, exactly as described in the *DESeq2* vignette. For example, the basic analysis would be run using the commands:

```
dds <- DESeq(dds);
res <- results(dds);
res
```

12.2 Perform analysis with edgeR

edgeR [2] performs Differential expression analysis of RNA-seq and digital gene expression profiles with biological replication. It uses empirical Bayes estimation and either exact tests and/or generalized linear models, based on the negative binomial distribution. As with *DESeq2*, The count data produced by QoRTs can be used by edgeR.

edgeR requires slightly more manipulation to put the count data in a format that it expects:

```
suppressPackageStartupMessages(library(edgeR));

decoder.bySample <- read.table(
                    "inputData/annoFiles/decoder.bySample.txt",
                    header=T,stringsAsFactors=F);

directory <- "outputData/countTables/";
files <- paste0(directory,
                    decoder.bySample$sample.ID,
                    "/QC.geneCounts.formatted.for.DESeq.txt.gz"
                  );
countData <- lapply(files, function(f){
     ct <- read.table(f,header=F,stringsAsFactors=F)$V2;
     ct <- ct[1:(length(ct)-5)]
});

countMatrix <- do.call(cbind.data.frame,countData);
colnames(countMatrix) <- decoder.bySample$sample.ID;
rownames(countMatrix) <- read.table(files[1],header=F,
                         stringsAsFactors=F)$V1[1:nrow(countMatrix)]

group <- factor(decoder.bySample$group.ID);
```

From here, the data is ready for use with edgeR. edgeR can be run as shown in the "Quick Start" example in the edgeR user guide. For example, to perfom a standard analysis based on generalized linear models:

```
design <- model.matrix(~group)
y <- DGEList(counts = countMatrix, group = group);
y <- calcNormFactors(y)
y <- estimateGLMCommonDisp(y,design)
y <- estimateGLMTrendedDisp(y,design)
y <- estimateGLMTagwiseDisp(y,design)
fit <- glmFit(y,design)
lrt <- glmLRT(fit,coef=2)
topTags(lrt)
```

12.3 Perform analysis with DEXSeq

DEXSeq [3] is another bioconductor-based tool designed to detect "differential exon usage" (DEU), as a proxy for detecting differentials in transcript-level expression regulation. QoRTs produces count files designed specifically for use with *DEXSeq* as part of its initial processing QC run (see section 6). In addition, *DEXSeq* requires a special "flattened" gff file, which QoRTs is also capable of generating (see section 10.1). IMPORTANT NOTE: the count files and the flattened gff file MUST be generated using the same strandedness mode (ie. with or without the `--stranded` option).

The DEXSeq-formatted data will use the file name `"QC.exonCounts.formatted.for.DEXSeq.txt.gz"`. The data can be loaded into *DEXSeq* using commands almost identical to those found in the *DEXSeq* package vignette:

```
suppressPackageStartupMessages(library(DEXSeq));
decoder <- read.table(
                    "inputData/annoFiles/decoder.bySample.txt",
                    header=T,stringsAsFactors=F);

directory <- "outputData/countTables/";

countFiles <- paste0(
               directory,
               decoder$sample.ID,
               "/QC.exonCounts.formatted.for.DEXSeq.txt.gz"
         );

dexseq.gff <- "outputData/forDEXSeq.gff.gz";

sampleTable <- data.frame(
     row.names = decoder$sample.ID,
     condition = decoder$group.ID
);

dxd <- DEXSeqDataSetFromHTSeq(
          countFiles,
          sampleData = sampleTable,
          design = ~sample + exon + condition:exon,
          flattenedfile=dexseq.gff
);
```

From here, any desired *DEXSeq* analyses can be run, as described in the package vignette.

For example, the basic DEU analysis can be run using the commands:

```
dxd <- estimateSizeFactors(dxd)
dxd <- estimateDispersions(dxd)
dxd <- testForDEU(dxd);
```

```
dxd <- estimateExonFoldChanges(dxd, fitExpToVar = "condition");
dxres <- DEXSeqResults(dxd);
dxres
```

13 References

[1] Simon Anders and Wolfgang Huber. Differential expression analysis for sequence count data. *Genome Biology*, 11:R106, 2010. URL: http://genomebiology.com/2010/11/10/R106.

[2] Mark D. Robinson and Gordon K. Smyth. Moderated statistical tests for assessing differences in tag abundance. *Bioinformatics*, 23:2881, 2007. URL: http://bioinformatics.oxfordjournals.org/cgi/content/abstract/23/21/2881, doi:10.1093/bioinformatics/btm453.

[3] Simon Anders, Alejandro Reyes, and Wolfgang Huber. Detecting differential usage of exons from RNA-seq data. *Genome Research*, 22:2008, 2012. doi:10.1101/gr.133744.111.

[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. URL: http://bioinformatics.oxfordjournals.org/content/29/1/15.abstract, doi:10.1093/bioinformatics/bts635.

[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. URL: http://bioinformatics.oxfordjournals.org/content/26/7/873.abstract, doi:10.1093/bioinformatics/btq057.

[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. URL: http://genomebiology.com/2013/14/4/R36, http://dx.doi.org/10.1186/gb-2013-14-4-r36 doi:10.1186/gb-2013-14-4-r36.

14 Session Information

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

```
sessionInfo()
```

```
## R version 3.1.1 (2014-07-10)
## Platform: x86_64-unknown-linux-gnu (64-bit)
##
## locale:
## [1] C
##
```

```
## attached base packages:
## [1] methods   stats     graphics  grDevices utils     datasets  base
##
## other attached packages:
## [1] QoRTs_0.2.11 Cairo_1.5-6  knitr_1.7
##
## loaded via a namespace (and not attached):
## [1] BiocStyle_1.4.1 evaluate_0.5.5  formatR_1.0     highr_0.4
## [5] stringr_0.6.2   tools_3.1.1
```

15 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
```