The QoRTs Analysis Pipeline Example Walkthrough

Stephen Hartley
National Human Genome Research Institute
National Institutes of Health

July 13, 2016

QoRTs v1.1.8
JunctionSeq v1.3.4

Contents

1 Overview

The latest version of this walkthrough will always be available here.

This walkthrough provides step-by-step instructions describing an analysis pipeline from post-alignment through QC, analysis, and the generation of summary visualizations for interpretation. Data processing and quality control is performed by the QoRTs software package, differential gene expression (DGE) analysis can be run via either DESeq2 or edgeR, and differential splicing analysis can be run using JunctionSeq or DEXSeq.

This walkthrough centers around the QoRTs [5] software package, which 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).

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.

The most recent release of QoRTs is available on the QoRTs github page.

The latest version of this walkthrough is available online, along with a full example dataset (file is 200mb) with example bam files (file is 1.1gb).

2 Using this Walkthrough

This walkthrough demonstrates how the use of this pipeline 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 `softwareRelease/QoRTs.jar`).

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

3 Requirements

Hardware: The QoRTs [5] 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 QoRTs 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. In general, it is preferable to run bioinformatic analysis on a dedicated linux-based cluster running some sort of job-queuing engine.

Software: The QoRTs software package requires R version 3.0.2 or higher, as well as java 6 or higher. It does not require any other software. Some of the other software packages used in this walkthrough may have their own individual dependencies, which may be subject to change over time.

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.

[1] Such as `DESeq`, `DESeq2` [1] or `edgeR` [8]

[2] Such as `DEXSeq` [2] or `JunctionSeq`

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

[4] See the gtf file specification here

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, 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 12.1.1 for details on the `--readGroup` option).

3.2 Additional Software Used In This Walkthrough

This walkthrough describes the use of a number of different software packages.

- **Aligners**: There are a number of aligners available for aligning RNA-Seq data. Using aligners other than RNA-STAR may require slight modification to the intial QoRTs data processing run, see the QoRTs FAQ for more details.
  - RNA-STAR [3] (Recommended)
  - TopHat2 [6]
  - GSNAP [9]
- **Differential Gene Expression Analysis (DGE):** These tools detect and assess differential expression at the gene level.
  - DESeq2 [1, 7] (Recommended)
  - edgeR [8]
- **Differential Splicing / Differential Isoform Regulation Analysis:** These tools detect and assess differential regulation of splice junctions and/or exonic regions as a proxy for differentials in isoform-level expression.
  - JunctionSeq (Our tool)
  - DEXSeq [2]
- **Other Tools**
  - QoRTs [5]: A Quality control and data processing utility.

- samtools: A suite of data processing tools.
- Novosort: A fast and efficient tool for sorting BAM files.
- The UCSC genome browser: A viewer that allows browsing of genome-wide sequence data in various forms.
- UCSC tools: Various tools for use with genomic data. In this walkthrough we use the tools `bedToBigBed` `wigToBigWig` and `fetchChromSizes` to prep track files for use with UCSC genome browser trackhubs.

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 [5] is designed to run on single-ended or paired-ended next-gen RNA-Seq data. The data must be aligned (or "mapped") to a reference genome before QoRTs can be run. RNA-Star [3], GSNAP [9], 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.

We generally recommend RNA-STAR for most purposes.

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. A few genes had additional fictional transcripts added, to test and demonstrate various tools' handling of certain edge cases. The original dataset from which these samples were derived is described elsewhere [4]. The original complete dataset is available on the NCBI Gene Expression Omnibus, series accession number GSE63309.

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 chromosome 14 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, X, M, and loose chromosome 14 contigs from the chrom.sizes file generated by the UCSC `fetchChromSizes` utility.
- `inputData/annoFiles/decoder.bySample.txt`: A sample decoder, which defines each sample's biological condition (CASE or CTRL).
- `inputData/annoFiles/decoder.byUID.txt`: 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 (file is 200mb) with example bam files (file is 1.1gb).

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: `QoRTsPipelineWalkthrough/`.

This example walkthrough will additionally the bioconductor packages: `DESeq2`, `edgeR`, and `DEXSeq`. Additionally, you will need to install the `JunctionSeq` R package and the `QoRTs` R companion package. It is recommended that you install the CRAN `png` package.

These packages can be installed with the R commands:

```
#RECOMMENDED CRAN package:
install.packages("png")

#REQUIRED bioconductor packages:
source("http://bioconductor.org/biocLite.R")
biocLite()
biocLite("DESeq2")
biocLite("DEXSeq")
biocLite("edgeR")

#Install JunctionSeq:
source("http://hartleys.github.io/JunctionSeq/install/JS.install.R");
JS.install();

#Install the QoRTs companion R Package:
install.packages("http://hartleys.github.io/QoRTs/QoRTs_LATEST.tar.gz",
                 repos=NULL,
                 type="source");
```

QoRTs R Package: It is generally preferable to download the most recent release of QoRTs, as shown above. However the version of QoRTs used to generate the example output data contained in this walkthrough is included with the example dataset. This version of QoRTs can be installed using the console command:

```
R CMD INSTALL softwareRelease/QoRTs_<versionNumber>.tar.gz
```

Again, this command must be executed from the root directory of the example dataset (`QoRTsPipelineWalk`

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
```

6 Initial Data Processing (STEP 1)

The first step is to process the aligned RNA-Seq data using QoRTs [5]. 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 softwareRelease/QoRTs.jar QC \
                    --stranded \
                    --chromSizes inputData/annoFiles/chrom.sizes \
                    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/$line/
  java -Xmx1G -jar softwareRelease/QoRTs.jar \
                QC \
                --stranded \
                --chromSizes inputData/annoFiles/chrom.sizes \
                inputData/bamFiles/$line.bam \
                inputData/annoFiles/anno.gtf.gz \
                outputData/qortsData/$line/
done < "inputData/annoFiles/uniqueID.list.txt"
```

Note that the `--chromSizes` option is optional, but causes QoRTs to additionally generate "wiggle" coverage files for each replicate. Note that this also increases the memory usage of QoRTs considerably. If you are having memory-usage issues, you may want to eliminate this option.

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

```
java -jar softwareRelease/QoRTs.jar QC --man
```

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

- *MAPQ filter:* Most aligners designed for RNA-Seq use the MAPQ field to define whether or not a read is multi-mapped. However, different aligners use different conventions. By default QoRTs assumes the RNA-STAR convention in which a MAPQ of 255 indicates a unique alignment. For TopHat, the `--minMAPQ` parameter must be set to 50. For GSNAP, the MAPQ behavior is not well documented (or at least I have been unable to find such documentation), but a MAPQ filtering threshold of 30 appears to work. See the FAQ for more information.
- *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 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 either name or position (under the default mode, it does not matter which). QoRTs can also be run in name-sorted-only mode using the "–nameSorted" option, which explicitly requires name-sorted data but may run faster and with lower memory overhead.
- *Single-end vs paired-end:* By default, QoRTs assumes the input bam file consists of paired-end data. For single-end data, the `--singleEnded` option must be used.
- *'Wiggle' Browser Tracks:* '.wig' browser tracks can also be produced in this step. See Section 12 for more information.
- *Junction Browser Tracks:* '.bed' junction depth browser tracks can also be produced in this step. See Section 12 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 Generating Counts and QC plots (Step 2)

Once the data has been processed, you can read the QC data produced by QoRTs [5] 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[5].

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:

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

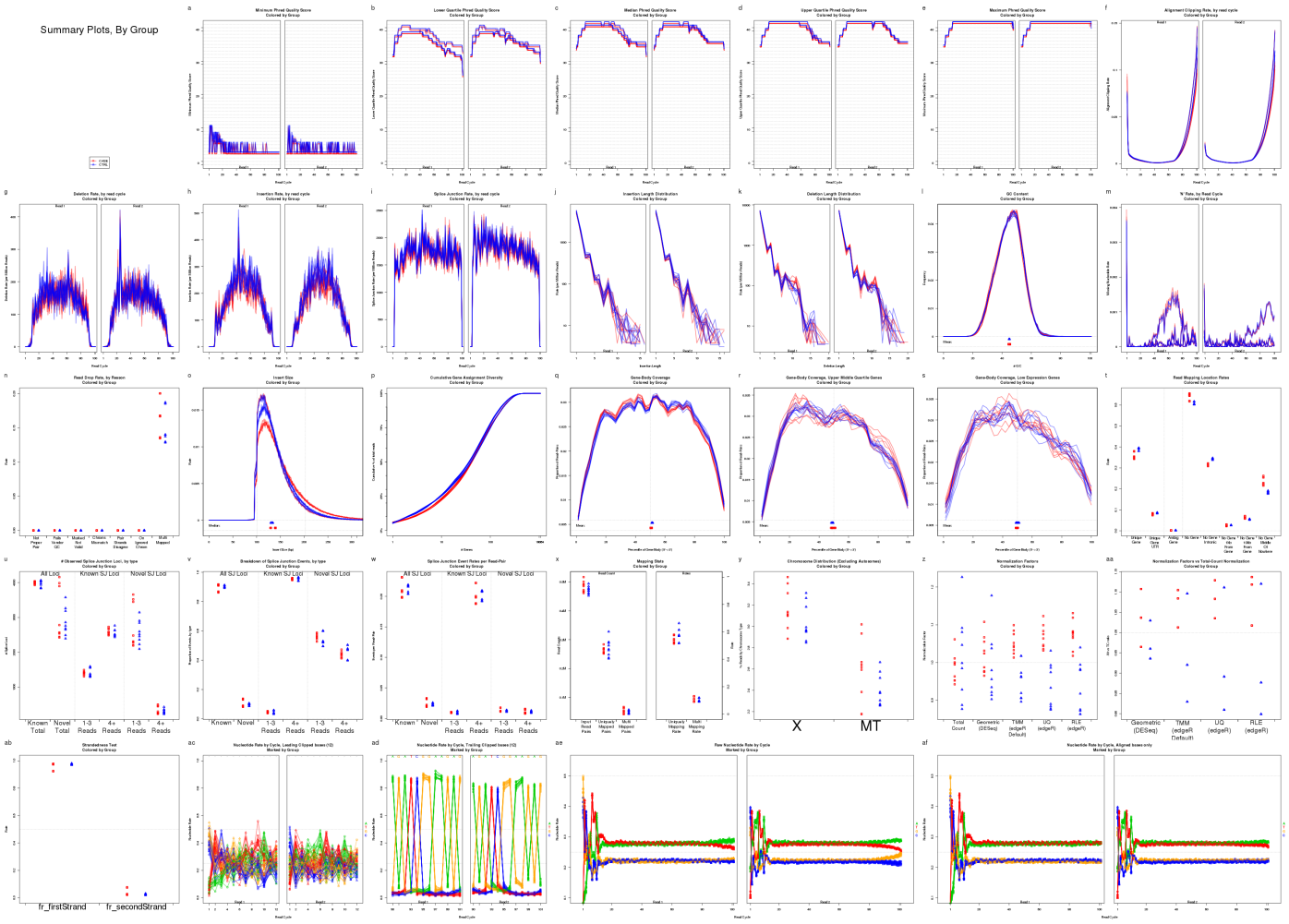

Figure 1: An example combined multi-plot. In this example, replicates are differentiated by color and pch symbol based on the group.ID (ie CASE or CTRL).

```
get.size.factors(res, outfile = "outputData/sizeFactors.GEO.txt");
```

8 Examine Data for Quality Control Issues (STEP 3)

Once the quality control plots have been generated, you must examine them to determine if any quality control issues do exist. For the example dataset these plots are available in the `outputData/qortsPlots/` directory.

QoRTs [5] produces a vast array of output data, and the interpretation of said data can be difficult. Proper quality control must consider the study design, sequencer technology, study species, read length, library preparation protocol, and numerous other factors that might affect the metrics produced by QoRTs. In some datasets, apparent "abnormalities" may be expected. Similarly, depending on the type of downstream analysis that is being performed, some errors or artifacts may be irrelevant.

When an unexplained abnormality is recognized, one must decide what to do with the data. Unfortunetely this question is nontrivial, and depends on numerous factors. Bioinformaticians must be aware of the statistical assumptions that are being made in the downstream analyses, and must consider the conditions under which such assumptions would be violated.

Some abnormalities will not affect a given analysis and thus can be ignored outright. Some may require that the offending sample(s) be removed from the analysis. Others may necessitate additional steps to normalize the data or adjust for confounding factors. And finally, some artifacts may be so severe that the intended analysis will be impossible using the supplied data. Ultimately bioinformaticians must use their own judgement to determine what action should be taken, should an abnormality be discovered

See Section 7 and 8 of the QoRTs vignette for more information.

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 [5] 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 softwareRelease/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 12.2. 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 softwareRelease/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  softwareRelease/QoRTs.jar mergeCounts --man
```

and

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

10 Perform Differential Gene Expression Analyses (STEP 5)

QoRTs [5] 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.

10.1 Perform DGE 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 12.1.1), 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
```

10.2 Optional Alternative: Perform DGE analysis with edgeR

`edgeR` [8] 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)
```

11 Perform Differential Splicing Analyses (STEP 6)

In addition to differential gene expression which tests entire genes for differential expression, some genes may exhibit isoform-level differential regulation. In these genes, individual isoforms (or "transcripts") may be differentially up or down regulated. This may occur with or without differential regulation of the other isoforms or of the gene as a whole.

There are a number of tools that can be used for detecting such differentials.

11.1 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  softwareRelease/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  softwareRelease/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  softwareRelease/QoRTs.jar makeFlatGff --man
```

11.1.1 Add Novel Junctions

The initial JunctionSeq input files produced by QoRTs only include annotated splice junctions. QoRTs includes all novel splice junctions, but they are counted seperately and not assigned unique cross-sample identifiers. It may be desirable to visually inspect or test 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 softwareRelease/QoRTs.jar \
                 mergeNovelSplices  \
                 --minCount 6 \
                 --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  softwareRelease/QoRTs.jar mergeNovelSplices --man
```

11.2 Perform Analysis with JunctionSeq

We have developed the JunctionSeq R package, which operates on the same basic principles as the DEXSeq bioconductor package but can also test for differential usage of known or novel splice junctions. Details and documentation for the JunctionSeq package are available online. For a more in-depth description of how to use JunctionSeq, see the JunctionSeq vignette.

JunctionSeq can be installed with the simple command:

```
source("http://hartleys.github.io/JunctionSeq/install/JS.install.R");
JS.install();
```

This calls a simple helper script that installs all prerequisite packages and then installs the JunctionSeq package itself. If you encounter any problems you may need to install JunctionSeq manually. See the JunctionSeq FAQ for more information.

11.2.1 Simple analysis pipeline

JunctionSeq includes a single function that loads the count data and performs a full analysis automatically. This function internally calls a number of sub-functions and returns a JunctionSeqCountSet object that holds the analysis results, dispersions, parameter estimates, and size factor data.

```
#Load JunctionSeq:
library("JunctionSeq");
#The sample decoder:
decoder <- read.table(
                    "inputData/annoFiles/decoder.bySample.txt",
                    header=T,stringsAsFactors=F);
#The count files:
countFiles <- paste0(
               "outputData/countTables/",
               decoder$sample.ID,
               "/QC.spliceJunctionAndExonCounts.withNovel.forJunctionSeq.txt.gz"
         );

#Run the analysis:
jscs <- runJunctionSeqAnalyses(
    sample.files = countFiles,
    sample.names = decoder$sample.ID,
    condition = decoder$group.ID,
    flat.gff.file = "outputData/countTables/withNovel.forJunctionSeq.gff.gz",
    nCores = 1,
    verbose=TRUE,
    debug.mode = TRUE
    );
```

The default analysis type, which is explicitly set in the above command, performs a "hybrid" analysis that tests for differential usage of both exons *and* splice junctions simultaniously. The methods used to test for differential splice junction usage are extremely similar to the methods used for differential exon usage other than the fact that the counts are derived from splice sites rather than exonic regions. These methods are based on the methods used by the DEXSeq package [2, 7].

The advantage to our method is that reads (or read-pairs) are never counted more than once in any given statistical test. This is not true in DEXSeq, as the long paired-end reads produced by current sequencers may span several exons and splice junctions and thus be counted several times. Our method also produces estimates that are more intuitive to interpret: our fold change estimates are defined as the fold difference between two conditions across a transcript sub-feature relative to the fold change between the same two conditions for the gene as a whole. The DEXSeq method instead calculates a fold change relative to the sum of the other sub-features, which may change depending on the analysis inclusion parameters.

The above function has a number of optional parameters, which may be relevant to some analyses:

`analysis.type` : By default JunctionSeq simultaniously tests both splice junction loci and exonic regions for differential usage (a "hybrid" analysis in which both exons and splice junctions are tested). This parameter can be used to limit analyses specifically to either splice junction loci or exonic regions.

`nCores` : The number of cores to use (note: this will only work when package BiocParallel is used on a Linux-based machine. Unfortunately R cannot run multiple threads on windows at this time).

`meanCountTestableThreshold` : The minimum normalized-read-count threshold used for filtering out low-coverage features.

`test.formula0` : The model formula used for the null hypothesis in the ANODEV analysis.

`test.formula1` : The model formula used for the alternative hypothesis in the ANODEV analysis.

`effect.formula` : The model formula used for estimating the effect size and parameter estimates.

`geneLevel.formula` : The model formula used for estimating the gene-level expression.

A full description of all these options and how they are used can be accessed using the command:

```
help(runJunctionSeqAnalyses);
```

11.2.2 Extracting test results

Once the differential splice junction usage analysis has been run, the results, including model fits, fold changes, p-values, and coverage estimates can all be written to file using the command:

```
writeCompleteResults(jscs,
                     outfile.prefix="outputData/analyses/JunctionSeq/",
                     save.jscs = TRUE
                     );
```

This produces a series of output files. The main results files are allGenes.results.txt.gz and sigGenes.results.t The former includes rows for all genes, whereas the latter includes only genes that have one or more statistically significant differentially used splice junction locus. The columns for both are:

- *featureID*: The unique ID of the exon or splice junction locus.
- *geneID*: The unique ID of the gene.
- *countbinID*: The sub-ID of the exon or splice junction locus.
- *testable*: Whether the locus has sufficient coverage to test.
- *dispBeforeSharing*: The locus-specific dispersion estimate.
- *dispFitted*: The the fitted dispersion.
- *dispersion*: The final dispersion used for the hypothesis tests.
- *pvalue*: The raw p-value of the test for differential usage.
- *padjust*: The adjusted p-value, adjusted using the "FDR" method.
- *chr, start, end, strand*: The (1-based) positon of the exonic region or splice junction.
- *transcripts*: The list of known transcripts that contain this splice junction. Novel splice junctions will be listed as "UNKNOWN_TX".
- *featureType*: The type of the feature (exonic region, novel splice junction or known splice junction)
- *baseMean*: The base mean normalized coverage counts for the locus across all conditions.
- *HtestCoef(A/B)*: The interaction coefficient from the alternate hypothesis model fit used in the hypothesis tests. This is generally not used for anything but may be useful for certain testing diagnostics.

- *log2FC(A/B)*: The estimated log2 fold change found using the effect-size model fit. This is calculated using a different model than the model used for the hypothesis tests.
- *log2FCvst(A/B)*: The estimated log2 fold change between the vst-transformed estimates. This can be superior to simple fold-change estimates for the purposes of ranking effects by their effect size. Simple fold-change estimates can be deceptively high when expression levels are low.
- *expr_A*: The estimate for the mean normalized counts of the feature in group A.
- *expr_B*: The estimate for the mean normalized counts of the feature in group B.
- *geneWisePadj*: The gene-level q-value, for the hypothesis that one or more features belonging to this gene are differentially used. This value will be the same for all features belonging to the same gene.

Note: If the biological condition has more than 2 categories then there will be multiple columns for the HtestCoef and log2FC. Each group will be compared with the reference group. If the supplied condition variable is supplied as a `factor` then the first "level" will be used as the reference group.

There will also be a file labelled "sigGenes.genewiseResults.txt.gz", which includes summary information for all statistically-significant genes:

- *geneID*: The unique ID of the gene.
- *chr, start, end, strand*: The (1-based) positon of the exonic region or splice junction.
- *baseMean*: The base mean normalized coverage counts for the gene across all conditions.
- *geneWisePadj*: The gene-level q-value, for the hypothesis that one or more features belonging to this gene are differentially used. This value will be the same for all features belonging to the same gene.
- *mostSigID*: The sub-feature ID for the most significant exon or splice-junction belonging to the gene.
- *mostSigPadjust*: The adjusted p-value for the most significant exon or splice-junction belonging to the gene.
- *numExons*: The number of (known) non-overlapping exonic regions belonging to the gene.
- *numKnown*: The number of known splice junctions belonging to the gene.
- *numNovel*: The number of novel splice junctions belonging to the gene.
- *exonsSig*: The number of statistically significant non-overlapping exonic regions belonging to the gene.
- *knownSig*: The number of statistically significant known splice junctions belonging to the gene.
- *novelSig*: The number of statistically significant novel splice junctions belonging to the gene.
- *numFeatures*: The columns numExons, numKnown, and numNovel, separated by slashes.
- *numSig*: The columns exonsSig, knownSig, and novelSig, separated by slashes.

The writeCompleteResults function also writes a file: allGenes.expression.data.txt.gz. This file contains the raw counts, normalized counts, expression-level estimates by condition value (as normalized read counts), and relative expression estimates by condition value. These are the same values that are plotted in Section 11.2.3.

Finally, this function also produces splice junction track files suitable for use with the UCSC genome browser or the IGV genome browser. These files are described in detail in Section 12.1.2. If the save.jscs parameter is set to TRUE, then it will also save a binary representation of the JunctionSeqCountSet object.

11.2.3 Visualization and Interpretation

The interpretation of the results is almost as important as the actual analysis itself. Differential splice junction usage is an observed phenomenon, not an actual defined biological process. It can be caused by a number of underlying regulatory processes including alternative start sites, alternative end sites, transcript truncation, mRNA destabilization, alternative splicing, or mRNA editing. Depending on the quality of the transcript annotation, apparent differential splice junction usage can even be caused by simple gene-level differential expression, if (for example) a known gene and an unannotated gene overlap with one another but are independently regulated.

Many of these processes can be difficult to discern and differentiate. Therefore, it is imperative that the results generated by JunctionSeq be made as easy to interpret as possible.

JunctionSeq, especially when used in conjunction with the QoRTs software package, contains a number of tools for visualizing the results and the patterns of expression observed in the dataset.

```
buildAllPlots(
            jscs=jscs,
            FDR.threshold = 0.01,
            outfile.prefix = "outputData/analyses/JunctionSeq/results/",
            variance.plot = TRUE,
            ma.plot = TRUE,
            rawCounts.plot=TRUE,
            verbose = TRUE);
```

This will produce 4 subdirectories in the "plots" directory. Each sub-directory will contain 1 figure for each gene that contains one or more exons or junctions that achieves genome-wide statistical significance. Alternatively, plots for manually-specified genes can be generated using the gene.list parameter, which overrides the FDR.threshold parameter.

It will also produce an html file "testForDU.html", which will allow the user to browse and view all the plots produced by JunctionSeq.

11.3 Optional Alternative: Perform analysis with DEXSeq

`DEXSeq` [2] 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 12.1.1). In addition, `DEXSeq` requires a special "flattened" gff file, which QoRTs is also capable of generating (see section 11.1). 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 syntax used by DEXSeq has changed repeatedly between different versions, and may change again in the future. This walkthrough was written for use with DEXSeq version 1.12.1.

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
```

The DEU analysis results can then be written to file:

```
write.table(dxres, file = "outputData/analyses/DEXSeq/DEXSeq.results.txt");
```

12 Generating Browser Tracks (STEP 7)

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 coverage across both genomic regions and across splice junctions. JunctionSeq produces similar browser tracks using the GLM modeling to produce coverage estimates and p-values.

QoRTs can generate both single-sample or single-replicate tracks, or summary tracks that calculates normalized mean coverage information across multiple samples.

An example of the browser track based visualizations available via this pipeline can be found here. This browser session uses a browser trackhub described in Section **??**. The trackhub can be found here.

12.1 Individual-Sample Tracks

12.1.1 Make sample wiggle tracks

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

In this walkthrough, the replicate and sample wiggle tracks are already generated as part of the main data processing pipeline in Section and 9 respectively. However, these files can also be generated manually with the following command:

```
java -Xmx1G -jar softwareRelease/QoRTs.jar \
              bamToWiggle \
              --stranded \
              --windowSize 100 \
              --negativeReverseStrand \
              --includeTrackDefLine \
              --rgbColor 0,0,0 \
              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 softwareRelease/QoRTs.jar \
              bamToWiggle \
              --stranded \
              --windowSize 100 \
              --negativeReverseStrand \
              --includeTrackDefLine \
```

```
                --rgbColor 0,0,0 \
                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.

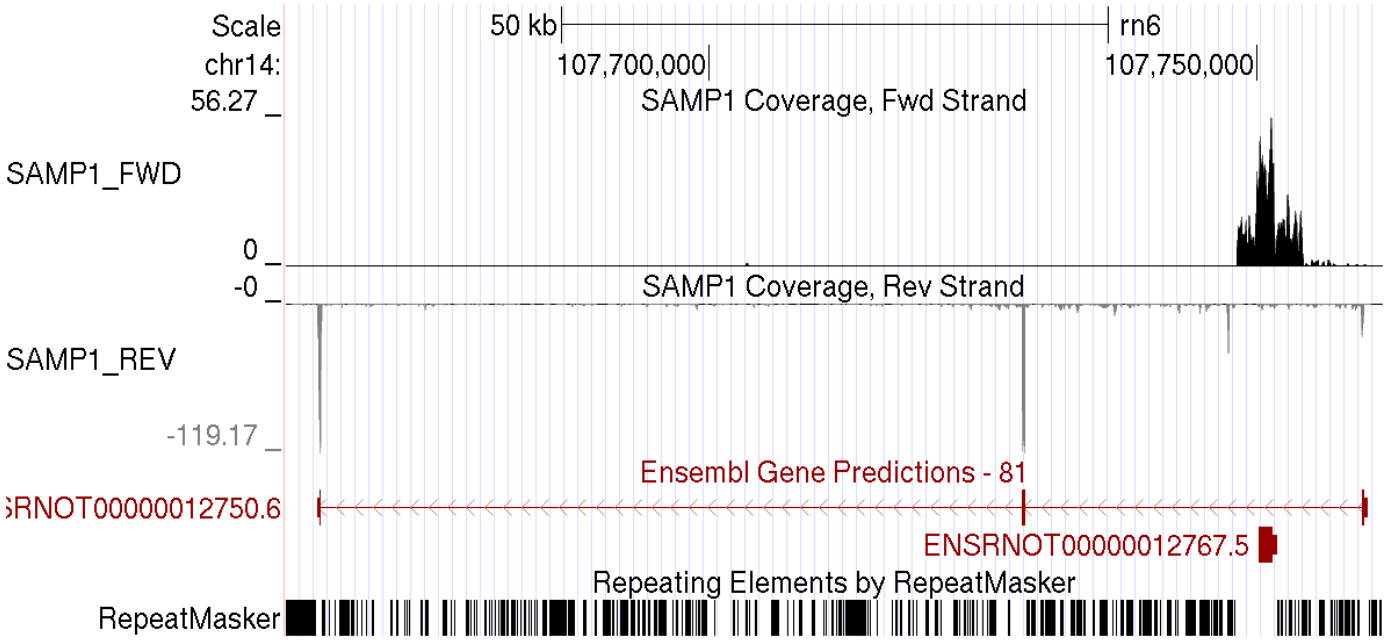

Figure 2: A pair of wiggle tracks generated for SAMP1. The top wiggle track plots the average coverage depth over the forward strand, the lower wiggle track plots the average coverage depth over the reverse strand. This track can be viewed on the UCSC browser here

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

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

It is generally easiest to generate the wiggle files during the initial processing step, as shown in Section 12.1.1). 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 softwareRelease/QoRTs.jar QC \
                    --stranded \
                    --chromSizes inputData/annoFiles/chrom.sizes \
                    --trackTitlePrefix SAMP1_RG1_WIGGLE  \
                    inputData/bamFiles/SAMP1_RG1.bam \
                    inputData/annoFiles/anno.gtf.gz \
                    outputData/qortsQcData/SAMP1_RG1/
```

12.1.2 Make sample junction tracks

QoRTs also includes utilities for creating splice junction coverage tracks for individual samples. See Figure 3 for an example, shown alongide wiggle tracks generated in Section 9.

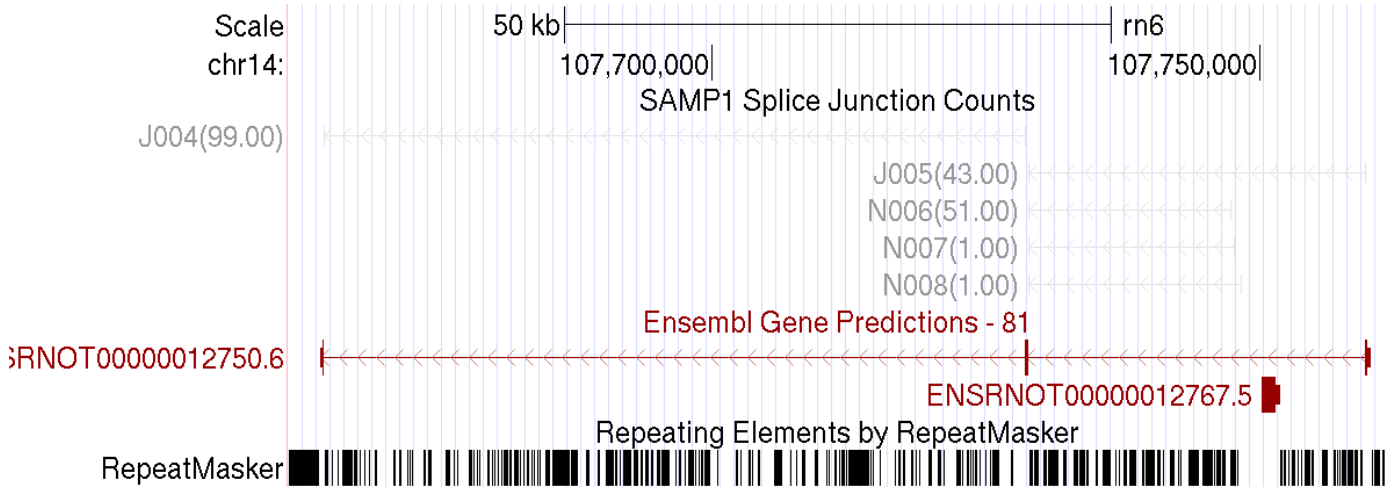

Figure 3: An example of the tracks that can be produced in this section. Each annotated splice junction locus has a identifier unique for each gene (example: J017). After the identifier, the number of reads (or read-pairs) that bridge the given splice junction locus is listed in parentheses. This track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

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

```
  java -Xmx1G -jar softwareRelease/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 softwareRelease/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  softwareRelease/QoRTs.jar makeJunctionTrack --man
```

12.1.3 Make sample junction tracks with novel junctions

Once the flattened annotation and count files which include the novel splice junctions has been generated (see Section 12.1.3), 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. Chimeric or cross-gene junctions will be ignored (but can still be found in the orphan junction gff file).

```
while read line
do
  java -Xmx1G -jar softwareRelease/QoRTs.jar \
       makeJunctionTrack \
       --filenames outputData/countTables/$line/QC.spliceJunctionAndExonCounts.wi\
thNovel.forJunctionSeq.txt.gz \
       outputData/countTables/withNovel.forJunctionSeq.gff.gz \
       outputData/countTables/$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  softwareRelease/QoRTs.jar makeJunctionTrack --man
```

12.2 Generate Summary Genome Tracks

While for small datasets it may be reasonable to visually inspect individual-sample browser tracks, for larger datasets this quickly becomes untenable. Thus, QoRTs includes functions for compiling multiple wiggle or junction-count browser tracks by normalized mean coverage. Note that this requires size factors for use in normalizing all the samples to a comparable scale. See Section for more details about how to extract size factors from the QoRTs output data. See Figure 4 for an example.

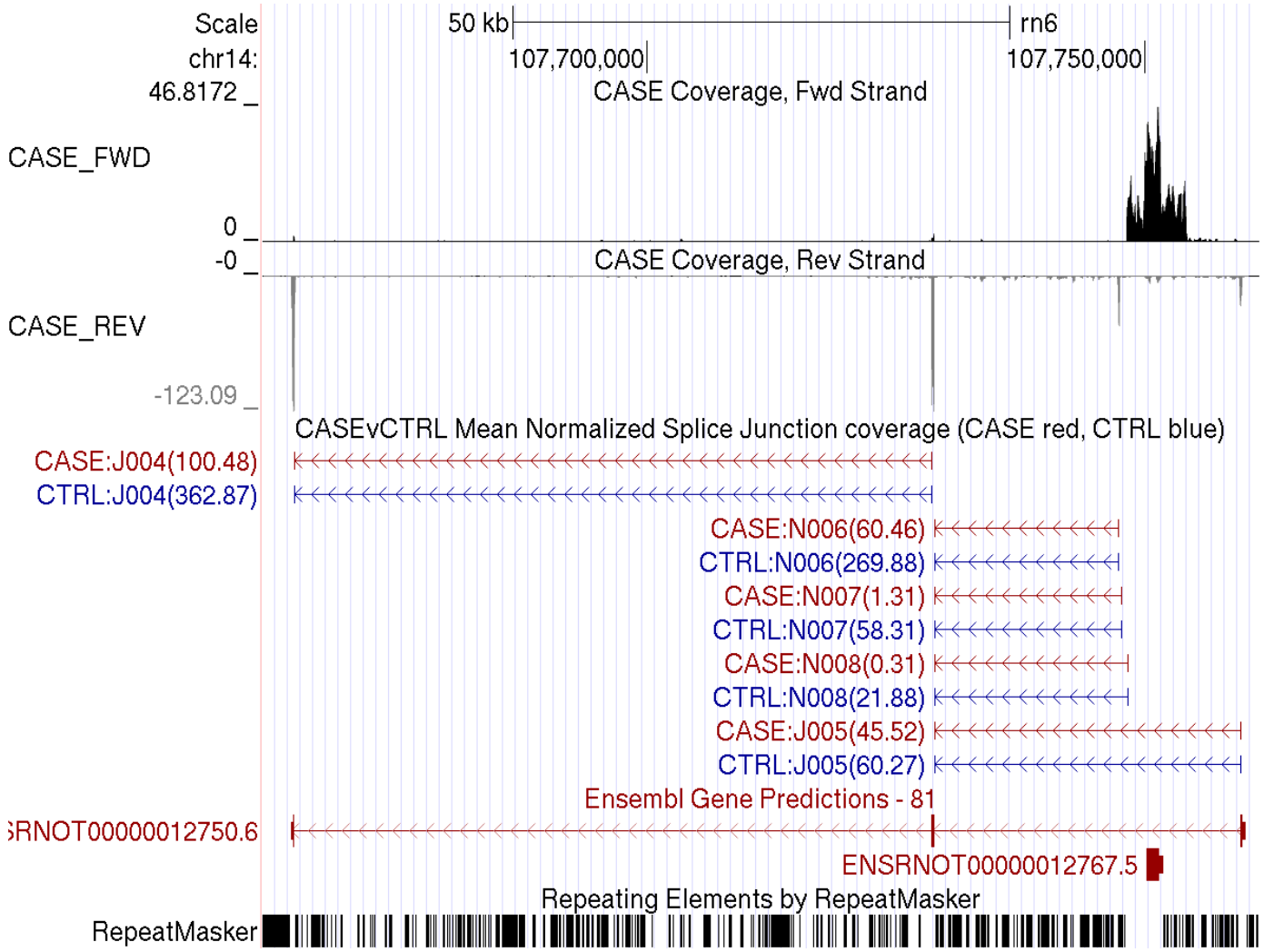

Figure 4: An example of the tracks that can be produced in Section 12.2. The first track ("CTRL_FWD") shows the mean normalized coverage depth for 100-base-pair windows for the forward (genomic) strand. The second track reveals the coverage depth on the negative strand (plotted as negative values). The 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. This track can be viewed on the UCSC browser here.

12.2.1 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 softwareRelease/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 softwareRelease/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 softwareRelease/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 softwareRelease/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  softwareRelease/QoRTs.jar mergeWig --man
```

12.2.2 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 softwareRelease/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 softwareRelease/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 12.1.3:

```
java -Xmx1G -jar softwareRelease/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  softwareRelease/QoRTs.jar makeJunctionTrack --man
```

12.2.3 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 softwareRelease/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 softwareRelease/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 on the first and second columns. We also need to remove the "track" lines from each file. In linux/bash this can be accomplished with the commands:

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

12.2.4 Make and Combine "orphan junction" tracks

By default, QoRTs and JunctionSeq only display splice junctions that span a single gene or group of overlapping genes. Novel splice junctions that bridge disjoint genes, or novel junctions that do not appear on any gene at all, are stored and counted separately as "ambiguous" and "orphan" junctions, respectively.

Just like with the "makeJunctionTrack" function, the "makeOrphanJunctionTrack" command can be used to create such summary plots. The only real difference is in the infileSuffix and the fact that this function does not require any GFF file as input.

For example, the following command will produce a junction track analagous to the first one generated in the previous section:

```
#Set cases to "red":
java -Xmx1G -jar softwareRelease/QoRTs.jar \
   makeOrphanJunctionTrack \
   --calcMean \
   --stranded \
   --rgb 150,0,0 \
   --title CASE \
   --infilePrefix outputData/countTables/ \
   --infileSuffix /QC.spliceJunctionCounts.orphanSplices.txt.gz \
   --sampleList SAMP1,SAMP2,SAMP3 \
   --sizeFactorFile outputData/sizeFactors.GEO.txt \
   outputData/mergedTracks/CASE.orphan.red.bed.gz

#And then set CONTROLS to "blue":
java -Xmx1G -jar softwareRelease/QoRTs.jar \
   makeOrphanJunctionTrack \
   --calcMean \
   --stranded \
   --rgb 0,0,150 \
   --title CTRL \
   --infilePrefix outputData/countTables/ \
   --infileSuffix /QC.spliceJunctionCounts.orphanSplices.txt.gz \
   --sampleList SAMP4,SAMP5,SAMP6 \
   --sizeFactorFile outputData/sizeFactors.GEO.txt \
   outputData/mergedTracks/CTRL.orphan.blue.bed.gz
```

These tracks can be generated and combined exactly as in Section 12.2.3:

```
zcat outputData/mergedTracks/CASE.orphan.red.bed.gz \
    outputData/mergedTracks/CTRL.orphan.blue.bed.gz | \
    grep -v "^track name" | \
    sort -k1,1 -k2,2n | gzip - > outputData/mergedTracks/CASE.vs.CTRL.orphan.bed.gz
```

12.2.5 Junction Tracks Produced by JunctionSeq

In addition to performing the analysis and generating output plots, JunctionSeq produces a number of browser tracks displaying summary information from the analysis.

The three main tracks are:

- allGenes.exonCoverage.bed: Displays all testable exonic regions, along with the GLM model estimates for the number of reads or read-pairs per sample for each biological condition.
- allGenes.junctionCoverage.bed: Displays all testable splice junctions, along with the GLM model estimates for the number of reads or read-pairs per sample for each biological condition.
- sigGenes.pvalues.bed: Shows the statistically significant loci and their adjusted p-values.

See 5 for an example of what these tracks look like. These files can be found in `outputData/analyses/Junct`

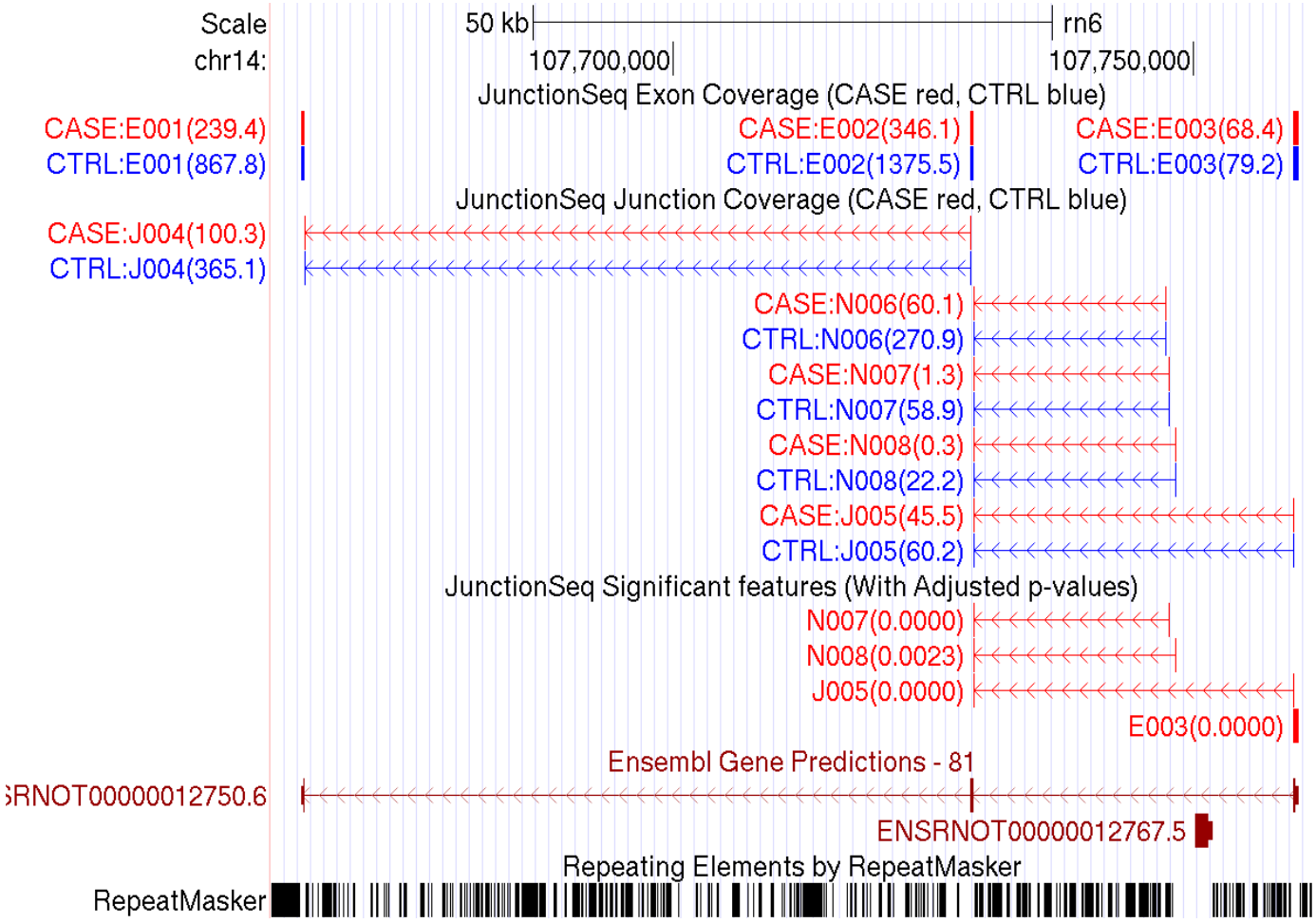

Figure 5: The three main browser tracks produced by JunctionSeq. These tracks can be viewed on the UCSC browser here

12.3 Advanced: UCSC Browser TrackHubs

See Figure 6 for an example of how these summary tracks can be viewed via TrackHubs. This walk-through only touches on a small fraction of the full capabilities of browser TrackHubs. For more complete information, see the UCSC track database definition (here).

With trackHubs, you can display multiple wiggle plots on top of one another, allowing easy visual comparisons between two or more conditions across the entire genome.

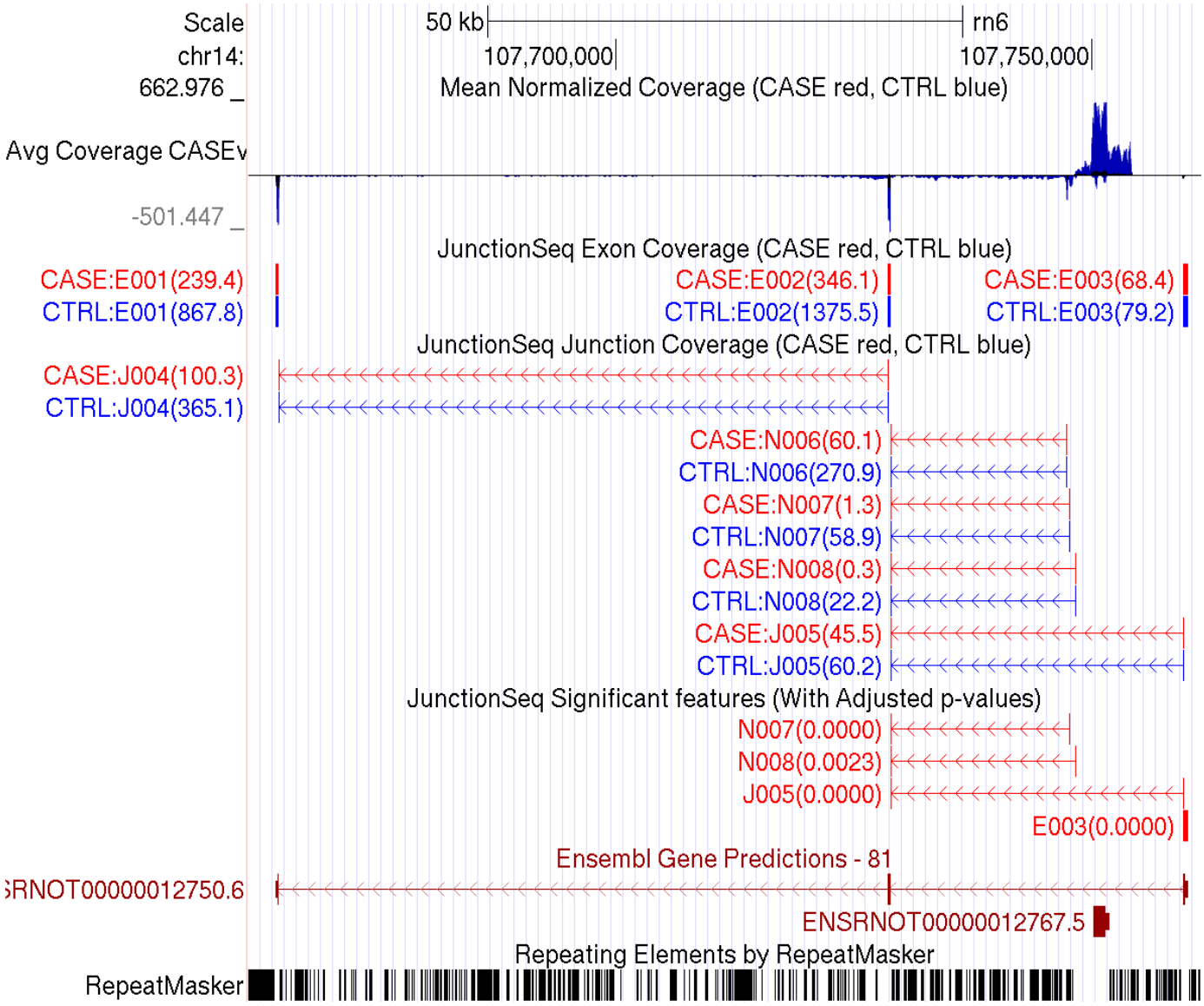

Figure 6: An example demonstrating how summary tracks can be viewed via the use of UCSC browser trackhubs. The aggregate "multiWig" track displays both cases and controls overlaid on top of one another in different colors. In addition, the forward and reverse strands are also overlaid, with the reverse strand displayed below zero. This track can be viewed on the UCSC browser here, and the hub itself can be found here.

Hosting a trackhub requires that you have somewhere to post data to the internet in such a way that it can be accessed by UCSC. If you do not have your own web host you can use a free service such as GitHub or dropbox.

12.3.1 Binary Conversion

TrackHub tracks require that data be converted into "binary" format, which is faster and more compact than the simpler text-based formats. This requires the UCSC tools `wigToBigWig`, `bedToBigBed`, and textttfetchChromSizes. These can be found online here, along with the other UCSC tools.

"WIG" files must be converted to "bigWig" files using the `wigToBigWig` utility:

```
wigToBigWig outputData/mergedTracks/CTRL.fwd.wig.gz \
            inputData/annoFiles/rn6.chrom.sizes \
            outputData/trackHub/rn6/tracks/CTRL.fwd.bw

wigToBigWig outputData/mergedTracks/CTRL.rev.wig.gz \
            inputData/annoFiles/rn6.chrom.sizes \
            outputData/trackHub/rn6/tracks/CTRL.rev.bw
```

Note that the chrom sizes file must generally be generated by the UCSC tool `fetchChromSizes`, or else errors may occur.

"BED" files must similarly be converted to "bigBed" files using the `bedToBigBed` utility. Note that this utility requires that the bed lines be sorted, and cannot process gzip-compressed files or files that begin with track or browser lines.

```
#Unzip the gzipped bed files, remove the track line and sort the results:
zcat outputData/analyses/JunctionSeq/allGenes.exonCoverage.bed.gz | \
     tail -n+2 | \
     sort -k1,1 -k2,2n > outputData/mergedTracks/JS/allGenes.exonCoverage.bed

#Convert this bed files into the binary "bigbed" format using the UCSC tool:
bedToBigBed outputData/mergedTracks/JS/allGenes.exonCoverage.bed \
            inputData/annoFiles/rn6.chrom.sizes \
            outputData/trackHub/rn6/tracks/JS.allGenes.exonCoverage.bb
```

This must be performed on all WIG and BED files that you wish to use with your trackHub.

12.3.2 Trackhub Setup

The trackhub consists of 3 control files: genomes.txt, hub.txt, and trackDB.txt. An example of these three files is included in the `inputData/trackHubFiles/` directory.

The full trackhub for the walkthrough data can be found here. A UCSC browser session with this trackhub loaded can be found here.

The "hub" file specifies the hub and tells the browser where to find the genomes file:

```
hub QoRTs_Pipeline_Example_Hub_v0.2.8
shortLabel QoRTs_Pipeline_Example_Hub_v0.2.8
longLabel QoRTs_Pipeline_Example_Hub_v0.2.8
genomesFile genomes.txt
email stephen.hartley
```

The "genomes" file specifies the genome and build, and directs the browser to the "trackDB" file:

```
genome rn6
trackDb rn6/trackDb.txt
```

The above two files must be in the main hub directory. There must also be a sub-directory with the same name as the genome.

Finally, inside the genomes directory you must place a trackDb file. This file actually controls what tracks to include and how they should be displayed. This file is similar to the "track" lines of standard UCSC browser track files, but with slightly different syntax. It allows the specification of "multiwig" tracks, which include multiple tracks plotted on top of one another on the same scale:

```
track CASEvCTRL_wiggle
container multiWig
aggregate transparentOverlay
showSubtrackColorOnUi on
shortLabel Avg Coverage CASEvCTRL
longLabel Mean Normalized Coverage (CASE red, CTRL blue)
boxedCgf on
autoScale on
visibility full
alwaysZero on
type bigWig

        track CTRL_FWD_SUB
        parent CASEvCTRL_wiggle
        type bigWig
        bigDataUrl trackFiles/CTRL.fwd.bw
        color 0,0,180
        altColor 0,0,180

        track CTRL_REV_SUB
        parent CASEvCTRL_wiggle
        type bigWig
        bigDataUrl trackFiles/CTRL.rev.bw
        color 0,0,180
        altColor 0,0,180

        track CASE_FWD_SUB
        parent CASEvCTRL_wiggle
        type bigWig
        bigDataUrl trackFiles/CASE.fwd.bw
        color 180,0,0
        altColor 180,0,0

        track CASE_REV_SUB
        parent CASEvCTRL_wiggle
        type bigWig
        bigDataUrl trackFiles/CASE.rev.bw
        color 180,0,0
        altColor 180,0,0
```

The example trackDb file can be found here.

13 The JctSeqData Package

A selection of the data generated and used in this walkthrough has been packaged and distributed in the `JctSeqData` R package. The data had to be reorganized in order to fit with the R package format. This section describes exactly how the JctSeqData package was generated.

First we copy over the template and add in the data generated in this walkthrough:

```
#make sure JctSeqData doesn't already exist:
rm -rf outputData/JctSeqData
#Copy over the template
cp -R inputData/JctSeqData-template outputData/JctSeqData
#Copy original annotation files:
cp inputData/annoFiles/*.* outputData/JctSeqData/inst/extdata/annoFiles/
#Copy additional generated annotation files:
cp outputData/forJunctionSeq.gff.gz                          \
   outputData/JctSeqData/inst/extdata/annoFiles/JunctionSeq.flat.gff.gz
cp outputData/forDEXSeq.gff.gz                               \
   outputData/JctSeqData/inst/extdata/annoFiles/DEXSeq.flat.gff.gz
cp outputData/countTables/orphanSplices.gff.gz               \
   outputData/JctSeqData/inst/extdata/annoFiles/
cp outputData/countTables/withNovel.forJunctionSeq.gff.gz \
   outputData/JctSeqData/inst/extdata/annoFiles/
#Copy count tables:
cp -R outputData/countTables/* outputData/JctSeqData/inst/extdata/cts/
```

Next we generate the "tiny" dataset used in the JunctionSeq examples and for rapid testing. This is done simply by using "egrep" to extract a subset of the genes from the various files:

```
cd outputData/JctSeqData/inst/extdata/
#Make a "egrep" regex string to extract the desired genes:
FILTER="ENSRNOG00000048600|ENSRNOG00000045591|etc. etc.";
#Note: this is only the start of the full regex string.
#      see file inputData/JctSeqData-template/inst/extdata/tinyGeneList.txt
#      for a full list of the extracted genes.

#Subsample annotation files:
zcat annoFiles/anno.gtf.gz | \
      egrep $FILTER - | \
      gzip -c - > tiny/anno.gtf.gz
zcat annoFiles/JunctionSeq.flat.gff.gz | \
      egrep $FILTER - | \
      gzip -c - > tiny/JunctionSeq.flat.gff.gz
zcat cts/withNovel.forJunctionSeq.gff.gz | \
      egrep $FILTER - | \
      gzip -c - > tiny/withNovel.forJunctionSeq.gff.gz
zcat cts/withNovel.forJunctionSeq.gff.gz | \
      egrep $FILTER - | \
      gzip -c - > tiny/withNovel.forJunctionSeq.gff.gz
#Subsample count files:
while read line
do
  mkdir ./tiny/$line
  echo $line
  zcat cts/$line/QC.spliceJunctionAndExonCounts.withNovel.forJunctionSeq.txt.gz | \
     egrep $FILTER - | \
     gzip -c - > tiny/$line/QC.spliceJunctionAndExonCounts.withNovel.forJunctionSeq.txt.gz
  zcat cts/$line/QC.exonCounts.formatted.for.DEXSeq.txt.gz | \
     egrep $FILTER - | \
     gzip -c - > tiny/$line/QC.exonCounts.formatted.for.DEXSeq.txt.gz
  zcat cts/$line/QC.geneCounts.formatted.for.DESeq.txt.gz | \
     egrep $FILTER - | \
     gzip -c - > tiny/$line/QC.geneCounts.formatted.for.DESeq.txt.gz
  zcat cts/$line/QC.spliceJunctionAndExonCounts.forJunctionSeq.txt.gz | \
     egrep $FILTER - | \
     gzip -c - > tiny/$line/QC.spliceJunctionAndExonCounts.forJunctionSeq.txt.gz
  zcat cts/$line/QC.spliceJunctionCounts.knownSplices.txt.gz | \
     egrep $FILTER - | \
     gzip -c - > tiny/$line/QC.spliceJunctionCounts.knownSplices.txt.gz
done < annoFiles/sampleID.list.txt
cd ../../../../
```

We can install this almost-finished version of the package using the command:

```
R CMD INSTALL outputData/JctSeqData
```

Next, we build the serialized "Rdata" files in R. To do this, first we load the datasets:

```
#Read the decoder:
decoder.file <- system.file("extdata/annoFiles/decoder.bySample.txt",
                            package="JctSeqData");
decoder <- read.table(decoder.file,
                      header=TRUE,
                      stringsAsFactors=FALSE);
#Here are the full-size count and gff files:
gff.file.FULL <- system.file("extdata/cts/withNovel.forJunctionSeq.gff.gz",
                        package="JctSeqData");
countFiles.FULL <- system.file(paste0("extdata/cts/",
      decoder$sample.ID,
      "/QC.spliceJunctionAndExonCounts.withNovel.forJunctionSeq.txt.gz"),
      package="JctSeqData");
#Here are the "tiny" subset count and gff files:
gff.file.TINY <- system.file("extdata/tiny/withNovel.forJunctionSeq.gff.gz",
                        package="JctSeqData");
countFiles.TINY <- system.file(paste0("extdata/tiny/",
      decoder$sample.ID,
      "/QC.spliceJunctionAndExonCounts.withNovel.forJunctionSeq.txt.gz"),
      package="JctSeqData");

geneID.to.symbol.file <- system.file(
                           "extdata/annoFiles/ensid.2.symbol.txt",
                           package="JctSeqData",
                           mustWork=TRUE);
```

Next, we generate the full dataset:

```
jscs2 <- runJunctionSeqAnalyses(sample.files = countFiles,
            sample.names = decoder$sample.ID,
            condition=factor(decoder$group.ID),
            flat.gff.file = gff.file,
            gene.names = geneID.to.symbol.file);
```

And save it:

```
save(jscs2, file = "outputData/JctSeqData/data/fullExampleDataSet.RData");
```

And then generate the "tiny" dataset:

```
jscs <- runJunctionSeqAnalyses(sample.files = countFiles.TINY,
            sample.names = decoder$sample.ID,
            condition=factor(decoder$group.ID),
            flat.gff.file = gff.file.TINY,
            gene.names = geneID.to.symbol.file);
```

And save it:

```
save(jscs, file = "outputData/JctSeqData/data/tinyExampleDataSet.RData");
```

14 References

References

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

[2] S. Anders, A. Reyes, and W. Huber. Detecting differential usage of exons from RNA-seq data. *Genome Research*, 22:2008, 2012.

[3] A. Dobin, C. A. Davis, F. Schlesinger, J. Drenkow, C. Zaleski, S. Jha, P. Batut, M. Chaisson, and T. R. Gingeras. STAR: ultrafast universal RNA-seq aligner. *Bioinformatics*, 29(1):15–21, 2013.

[4] S. W. Hartley, S. L. Coon, L. E. Savastano, J. C. Mullikin, C. Fu, D. C. Klein, and N. C. S. Program. Neurotranscriptomics: The effects of neonatal stimulus deprivation on the rat pineal transcriptome. *PloS ONE*, 10(9), 2015.

[5] S. W. Hartley and J. C. Mullikin. Qorts: a comprehensive toolset for quality control and data processing of rna-seq experiments. *BMC bioinformatics*, 16(1):224, 2015.

[6] D. Kim, G. Pertea, C. Trapnell, H. Pimentel, R. Kelley, and S. Salzberg. Tophat2: accurate alignment of transcriptomes in the presence of insertions, deletions and gene fusions. *Genome Biology*, 14(4):R36, 2013.

[7] S. A. Michael I Love, Wolfgang Huber. Moderated estimation of fold change and dispersion for rna-seq data with deseq2. *Genome Biology*, 15:550, 2014.

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

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

15 Legal

```
The MIT License
Copyright (c) 2009 The Broad Institute
Permission is hereby granted, free of charge, to any person
obtaining a copy of this software and associated documentation
files (the "Software"), to deal in the Software without
restriction, including without limitation the rights to use,
copy, modify, merge, publish, distribute, sublicense, and/or
sell copies of the Software, and to permit persons to whom the
Software is furnished to do so, subject to the following
conditions:
The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the Software.
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
OTHER DEALINGS IN THE SOFTWARE.
```

The MIT license and copyright information can also be accessed using the command:

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

JunctionSeq is based on the DEXSeq and DESeq2 packages, and is licensed with the GPL v3 license:

```
JunctionSeq is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

JunctionSeq is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with JunctionSeq.  If not, see <http://www.gnu.org/licenses/>.
```

Other software mentioned in this document are subject to their own respective licenses.