JunctionSeq Package User Manual

Stephen Hartley
National Human Genome Research Institute
National Institutes of Health

April 23, 2015

Contents

1 Overview

The JunctionSeq R package offers a powerful tool for testing differential junction usage (DJU) in next-generation, high-throughput RNA-Seq experiments. *Differential junction usage* is defined (for our purposes) as differences in the usage of splice junction sites relative to overall gene expression, associated with the experimental condition(s). This differential splice junction usage is used as a proxy for detecting isoform-level differentials such as isoform switching, alternative splicing, alternative start site usage, or alternative stop site usage.

JunctionSeq is *not* designed to detect changes in overall gene expression. Gene-level differential expression is best detected with tools designed specifically for that purpose such as DESeq2 [**?**] or edgeR [1].

2 Requirements

Software: JunctionSeq requires the QoRTs software package to produce the flattened annotation files and splice-junction count files necessary for analysis. The QoRTs software package requires R version 3.0.2 or higher, as well as java 6 or higher. JunctionSeq itself requires a number of R packages, which can be installed using the following R commands:

```
install.packages("statmod")
install.packages("plotrix")
install.packages("stringr")
source("http://bioconductor.org/biocLite.R")
biocLite()
biocLite("Biobase")
```

Additionally, multi-core execution can be enabled if and only if the BiocParallel bioconductor package is installed. This can be installed with the R commands:

```
source("http://bioconductor.org/biocLite.R")
biocLite()
biocLite("BiocParallel")
```

Hardware: Both the JunctionSeq and QoRTs software packages will generally require at least 4 gigabytes of RAM to run. In general at least 8gb is recommended if available.

Annotation: JunctionSeq requires a transcript annotation 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[1] suitable for these purposes. However, any format that adheres to the gtf file specification[2] will work.

Dataset: JunctionSeq requires aligned RNA-Seq data. Data can be paired-end or single-end, unstranded or stranded. It is strongly recommended, but not explicitly required, that the SAM/BAM files be sorted either by name or position (it does not matter which).

2.1 Alignment

QoRTs, which is used to generate read counts, is designed to run on paired-end or single-end next-gen RNA-Seq data. The data must first be aligned (or "mapped") to a reference genome. RNA-Star [2], GSNAP [3], and TopHat2 [4] are all popular and effective 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.

2.2 Recommendations

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.

This reduces "batch effects", reducing the chances of false discoveries being driven by sequencer artifacts or biases.

3 Example Dataset

To allow users to test JunctionSeq and experiment with its functionality, an example dataset is available online (ADD LINK).

[1] Which can be acquired from the Ensembl website

[2] See the gtf file specification (here)

The example dataset was taken from rat pineal glands. Sequence data from six samples (aka "biological replicates") are included, three harvested during the day, three at night. To reduce the file sizes to a more managable level, this dataset used only 3 out of the 6 sequencing lanes, and only the reads aligning to chromosome 14 were included. This yielded roughly 750,000 reads per sample. The example dataset, including aligned reads, QC data, example scripts, splice junction counts, and JunctionSeq results, is available online (see the JunctionSeq github page).

Splice junction counts and annotation files generated from this example dataset are included in the `JcnSeqExData` R package, available online (see the JunctionSeq github page), which is what will be used by this vignette.

The annotation files can be accessed with the commands:

```
decoder.file <- system.file("extdata/annoFiles/decoder.S.bySample.txt",
                            package="JctSeqExData",
                            mustWork=TRUE);
decoder <- read.table(decoder.file,
                      header=T,
                      stringsAsFactors=F);
gff.file <- system.file(
            "extdata/cts/withNovel.forJunctionSeq.gff.gz",
            package="JctSeqExData",
            mustWork=TRUE);

print(decoder);
```

```
##   sample.ID  group.ID subgroup.ID  TIME
## 1     SHAM1   ShamDay        Sham   Day
## 2     SHAM2   ShamDay        Sham   Day
## 3     SHAM3   ShamDay        Sham   Day
## 4     SHAM4 ShamNight        Sham Night
## 5     SHAM5 ShamNight        Sham Night
## 6     SHAM6 ShamNight        Sham Night
```

The count files can be accessed with the commands:

```
countFiles.noNovel <- system.file(paste0("extdata/cts/",
              decoder$sample.ID,
              "/QC.spliceJunctionAndExonCounts.forJunctionSeq.txt.gz"),
              package="JctSeqExData", mustWork=TRUE);

countFiles <- system.file(paste0("extdata/cts/",
              decoder$sample.ID,
              "/QC.spliceJunctionAndExonCounts.withNovel.forJunctionSeq.txt.gz"),
              package="JctSeqExData", mustWork=TRUE);
```

4 Preparations

Once alignment and quality control has been completed on the study dataset, the splice-junction and gene counts must be generated via QoRTs.

To reduce batch effects, the RNA samples used for the example dataset were barcoded and merged together into a single combined library. This combined library was run on three HiSeq 2000 sequencer lanes. Thus, after demultiplexing each sample consisted of three "technical replicates". JunctionSeq is only designed to compare biological replicates, so QoRTs includes functions for generating counts for each technical replicate and then combining the counts across the technical replicates from each biological sample.

4.1 Generating raw counts via QoRTs

To generate splice junction counts, you must run QoRTs on each aligned bam file. QoRTs includes a basic function that calculates a variety of QC metrics along with gene-level and splice-junction-level counts. All these functions can be performed in a single step and a single pass through the input alignment file, greatly simplifying the analysis pipeline.

For example, to run QoRTs on the first read-group of sample `S_D1` from the example dataset:

```
java -jar /path/to/jarfile/QoRTs.jar QC \
                   --stranded \
                   inputData/bamFiles/SHAM1_RG1.bam \
                   inputData/annoFiles/anno.gtf.gz \
                   rawCts/SHAM1_RG1/
```

Note that the `--stranded` option is required because this example dataset is strand-specific. Also note that QoRTs uses the original gtf annotation file, NOT the flattened gff file produced in section 4.3. More information on this command and on the available options can be found online here.

If Quality Control is being done seperately by other software packages or collaborators, the JunctionSeq counts can be generated alone by setting the `--runFunctions` option:

```
java -jar /path/to/jarfile/QoRTs.jar QC \
 --stranded \
 --runFunction writeKnownSplices,writeNovelSplices,writeSpliceExon \
 inputData/bamFiles/SHAM1_RG1.bam \
 inputData/annoFiles/anno.gtf.gz \
 rawCts/SHAM1_RG1/
```

This will take much less time to run, as it does not generate the full battery of quality control metrics.

For more information about the quality control metrics provided by QoRTs, and how to visualize, organize, and view them, see the QoRTs github page and documentation, available online here.

4.2 Merging Counts from Technical Replicates (If Needed)

QoRTs includes functions for merging all count data from various technical replicates. If your dataset does not include technical replicates, or if technical replicates have already been merged prior to the count-generation step then this step is unnecessary.

The example dataset has three such technical replicates per sample, which were aligned separately and counted separately. For the purposes of quality control QoRTs was run separately on each of these bam files (making it easier to discern any lane or run specific artifacts that might have occurred). It is then necessary to combine the read counts from each of these bam files.

QoRTs includes an automated utility for performing this merge. For the example dataset, the command would be:

```
java -jar /path/to/jarfile/QoRTs.jar \
                mergeAllCounts \
                rawCts/ \
                annoFiles/decoder.byUID.txt \
                cts/
```

the rawCts and cts are the relative paths to the input and output data, respectively. The "decoder" file should be a tab-delimited text file with column titles in the first row. One of the columns must be titled "sample.ID", and one must be labelled "unique.ID". The unique.ID must be unique and refers to the specific technical replicate, the sample.ID column indicates which biological sample each technical replicate belongs to.

4.3 (Option 1) Including Only Annotated Splice Junction Loci)

If you wish to only test annotated splice junctions, then a simple flat annotation file can be generated for use by JunctionSeq. This file parses the input gtf annotation and assigns unique identifiers to each feature (exon or splice junction) belonging to each gene. These identifiers will match the identifiers listed in the junction count files produced by QoRTs in the count-generation step (see Section 4.1).

```
java -jar /path/to/jarfile/QoRTs.jar makeFlatGtf \
                   --stranded \
                   annoFiles/anno.gtf.gz \
                   annoFiles/JunctionSeq.flat.gff.gz
```

Note it is *vitally important* that the same options and gtf annotation file are used for creating this flat gff file as were used in the count-generation step (described in Section 4.1)! If the counts are generated in stranded mode, the gff file must also be generated in stranded mode.

4.4 (Option 2) Including Novel Splice Junction Loci

One of the core advantages of JunctionSeq over similar tools such as DEXSeq is the ability to include novel (ie. unannotated) splice junctions. Most advanced aligners have the ability to align read-pairs to both known and unknown splice junctions. However, many of these splice junctions will only have one or two read pairs aligned across them. Many of these putative splice junctions may be artifacts caused by sequencing errors or mapping artifacts, and even if they are real JunctionSeq will not have the power to detect any differential splice junction usage across them. Therefore, it is generally desirable to first filter splice junctions by read depth.

In order to properly filter by read depth, size factors are needed. These can be generated in JunctionSeq (see section ADD REFERENCE), or generated from gene-level read counts using DESeq2 or edgeR.

```
java -jar /path/to/jarfile/QoRTs.jar QC \
                mergeNovelSplices  \
                --minCount 10 \
                --stranded \
                cts/ \
                sizeFactors.GEO.txt \
                annoFiles/anno.gtf.gz \
                cts/
```

This utility finds all splice junctions that fall inside the bounds of any known gene. It then filters this set of splice junctions, selecting only the junction loci with mean normalized read-pair counts of greater than the assigned threshold (set to 100 read-pairs in the example above). It then gives each splice junction that passes this filter a unique identifier.

This utility produces two sets of output files. First it writes a .gff file containing the unique identifiers for each annotated and novel-and-passed-filter splice locus. Secondly, for each sample it produces a merged splice junction count file listing the splice junction counts for each of these uniquely identified splice junction loci.

5 Testing for differential splice junction usage

5.1 Hypothesis testing

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 with analysis results, dispersions, parameter estimates, and size factor data. This function should be sufficient for most purposes.

```
jscs <- runJunctionSeqAnalyses(sample.files = countFiles,
                           sample.names = decoder$sample.ID,
                           condition=factor(decoder$group.ID),
                           flat.gff.file = gff.file,
                           nCores = 12
                           );
```

5.1.1 Advanced Analysis Pipeline

Some advanced users may need to deviate from the standard analysis pipeline. They may want to use different size factors, apply multiple different models without having to reload the data from file each time, or use other advanced features.

First you must create a "design" data frame:

```
design <- data.frame(condition = factor(decoder$group.ID));
```

Note: the experimental condition variable MUST be named "condition".

Next, the data must be loaded into a JunctionSeqCountSet:

```
jscs = readJunctionSeqCounts(countfiles = countFiles,
                             samplenames = decoder$sample.ID,
                             design = design,
                             flat.gff.file = gff.file
                             );
```

Next, size factors must be created and loaded into the dataset:

```
#Generate the size factors and load them into the JunctionSeqCountSet:
jscs <- estimateSizeFactors(jscs);
#Now print the size factors:
print(pData(jscs)$sizeFactor);
```

Next, we generate test-specific dispersion estimates:

```
jscs <- estimateJunctionSeqDispersions(jscs, nCores = 12);
```

Next, we fit these observed dispersions to a regression to create fitted dispersions:

```
jscs <- fitDispersionFunction(jscs);
```

Next, we perform the hypothesis tests for differential splice junction usage:

```
jscs <- testJunctionsForDJU(jscs, nCores = 12);
```

Finally, we calculate effect sizes and parameter estimates:

```
jscs <- estimateEffectSizes( jscs, nCores = 12);
```

All this functionality simply duplicates the behavior of the runJunctionSeqAnalyses function. However, far more options are available when run in this way. See the options for each of these commands using the commands:

```
help(readJunctionSeqCounts);
help(estimateSizeFactors);
help(estimateJunctionSeqDispersions);
help(fitDispersionFunction);
help(testJunctionsForDJU);
help(estimateEffectSizes);
```

5.2 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="./test"
                     );
```

This produces a series of output files. The main results files are allGenes.results.txt.gz and sigGenes.results.txt.gz. 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 splice locus.
- *geneID*: The unique ID of the gene.
- *countbinID*: The sub-ID of the splice 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 splice junction usage.
- *padjust*: The adjusted p-value, adjusted using the BH method.
- *chr, start, end, strand*: The (1-based) positon of the splice junction.
- *transcripts*: The list of known transcripts that contain this splice junction.
- *featureType*: the type of the feature (novel splice junction or known splice junction)
- *meanBase*: The base mean normalized coverage for the locus.
- *HtestCoef(A/B)*: The interaction coefficient from the alternate hypothesis model fit used in the hypothesis tests. This is generally not used for anything.
- *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.

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.

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 6.2.

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 6.3.3. If the save.jscs parameter is set to TRUE, then it will also save a binary representation of the JunctionSeqCountSet.

6 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 many are practically impossible to discern or even define in an automated fashion. Therefore it is imperative that the results generated by JunctionSeq be made as easy to interpret as possible.

JunctionSeq, 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.

6.1 Summary Plots

JunctionSeq provides functions for two basic summary plots, used to display experiment-wide results. The first is the dispersion plot, which displays the dispersion estimates (y-axis) as a function of the base mean normalized counts (x-axis). The test-specific dispersions are displayed as black dots, and the "fitted" dispersion is displayed as a red line. JunctionSeq can also produce an "MA" plot, which displays the fold change on the y-axis ("M") as a function of the overall mean normalized counts on the x-axis ("A").

These plots can be generated with the command:

```
plotDispEsts(jscs);
plotMA(jscs, FDR.threshold=0.05);
```

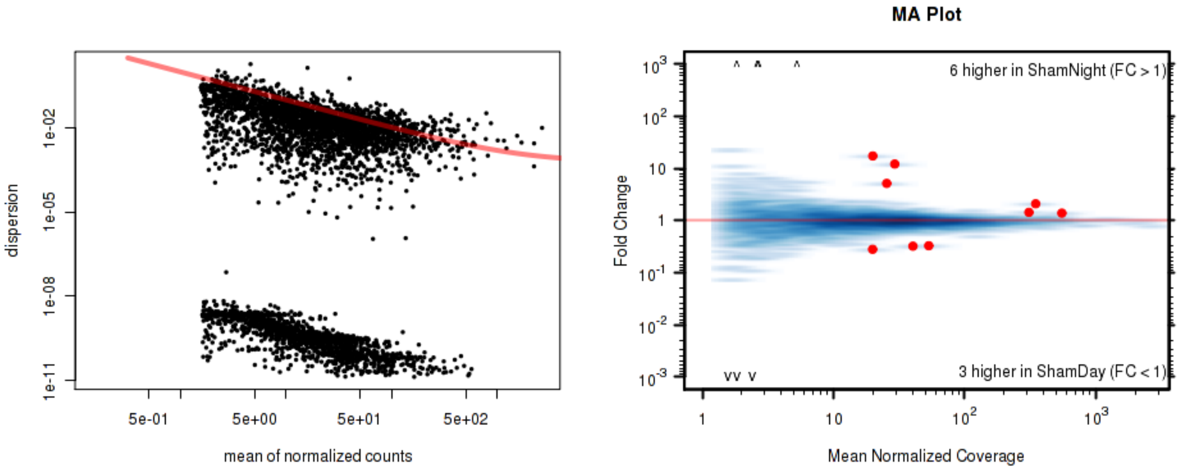

Figure 1: Summary Plots

6.2 Gene plots

JunctionSeq includes a simple and straightforward function that automatically generates a comprehensive battery of plots. Generally it is desired to print plots for all genes that have one or more splice junctions with statistically significant differential splice junction usage. By default, JunctionSeq uses an FDR-adjusted p-value threshold of 0.01.

These plots can be generated via the command:

```
buildAllPlots(jscs=jscs,
              flat.gff.file=gff.file,
              outfile.prefix = "./test",
              use.plotting.device = "png",
              FDR.threshold = 0.01);
```

This will produce 8 plots for each gene that includes differential splice junction usage. Alternatively, plots for manually-specified genes can be generated using the gene.list parameter, which overrides the FDR.threshold parameter.

For each selected gene, this function will produce 8 plots, two each of four plot types:

- *geneID-expr.png*: Estimates of average coverage depth for each biological condition. See Section 6.2.1.
- *geneID-rExpr.png*: Relative expression estimates for each biological condition (excluding gene-level differential effects). See Section 6.2.2.
- *geneID-rawCounts.png*: Raw read counts for each sample, colored by biological condition. See Section 6.2.3.
- *geneID-normCounts.png*: Normalized read counts for each sample, colored by biological condition. See Section 6.2.4.

Two versions of each plot are generated: one in which the full annotated transcript set is plotted beneath the expression graph, and one where only the simplified composite annotation is displayed.

6.2.1 Coverage/Expression Plots

Figure 2 displays the model estimates of the mean normalized splice junction coverage depth for each biological condition (in this example: cases vs controls). Note that these values are not equal to the simple mean normalized read counts across all samples. Rather, these estimates are derived from the GLM parameter estimates (via linear contrasts).

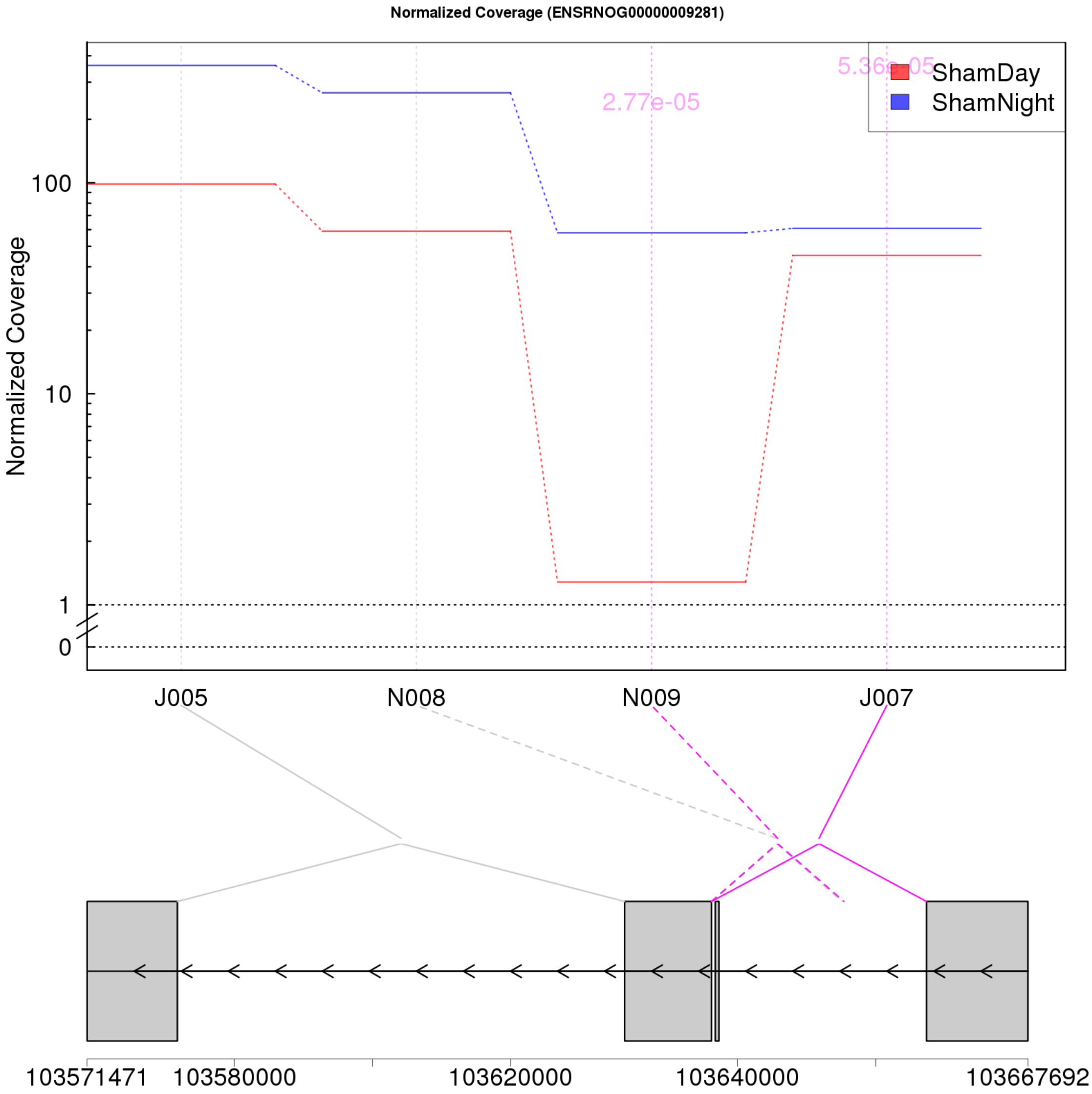

Figure 2: Splice junction coverage by biological condition. The plot includes 2 major frames. The top frame graphs the expression levels for each tested splice junction locus. Junctions that are statisically significant are marked with vertical pink lines, and the significant p-values are displayed along the top of the plot. The bottom frame is a line drawing of the known exons for the given gene, as well as all tested splice junctions. Known junctions are drawn with solid lines, novel junctions are dashed. Additionally, statistically significant junctions are colored pink. Between the two frames a set of lines connect the junction drawing to the expression plots.

There are a few things to note about this plot:

- The y-axis is log-transformed, except for the area between 0 and 1 which is plotted on a simple linear scale.
- In the line drawing at the bottom, the exons and introns are not drawn to a common scale. The exons are enlarged to improve readability. The rescaling is very simple: all exons are proportionately enlarged to take up 30 percent of the diagram. This can be adjusted via the "exon.rescale.factor" parameter (the default is 0.3), or turned off entirely by setting this parameter to -1.
- Note that junction J007 is marked as significantly differentially used, even though it is NOT differentially expressed. This is because JunctionSeq does not test for simple differential expression. It tests for differential splice junction coverage relative to gene-wide expression. Therefore, if (as in this case) the gene as a whole is strongly differentially expressed, then a splice junction that is NOT differentially expressed is the one that is being differentially used.
- Similarly, splice junction N009 is differentially used in the opposite way: while the gene itself is somewhat differentially expressed (at a fold change of roughly 3-4x overall), this particular junction has a *massive* differential, far beyond that found in the gene as a whole (at 45x fold change). Thus, it is being differentially used.

Figure 3 displays the same information found in Figure 2, except with all known transcripts plotted beneath the main plot.

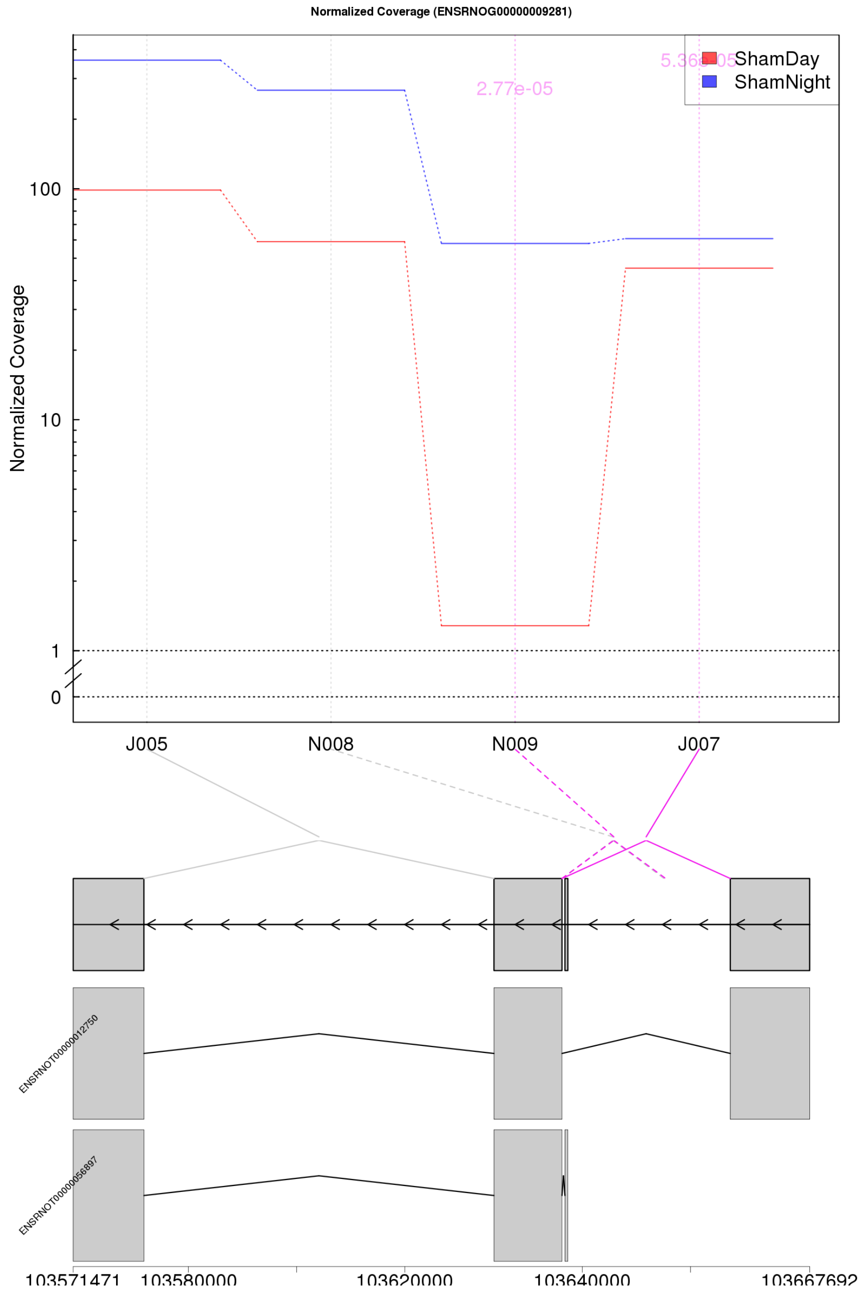

Figure 3: Splice junction coverage by biological condition, with annotated transcripts displayed. This plot is identical to the previous, except all annotated transcripts are displayed below the standard plot.

6.2.2 Relative Expression Plots

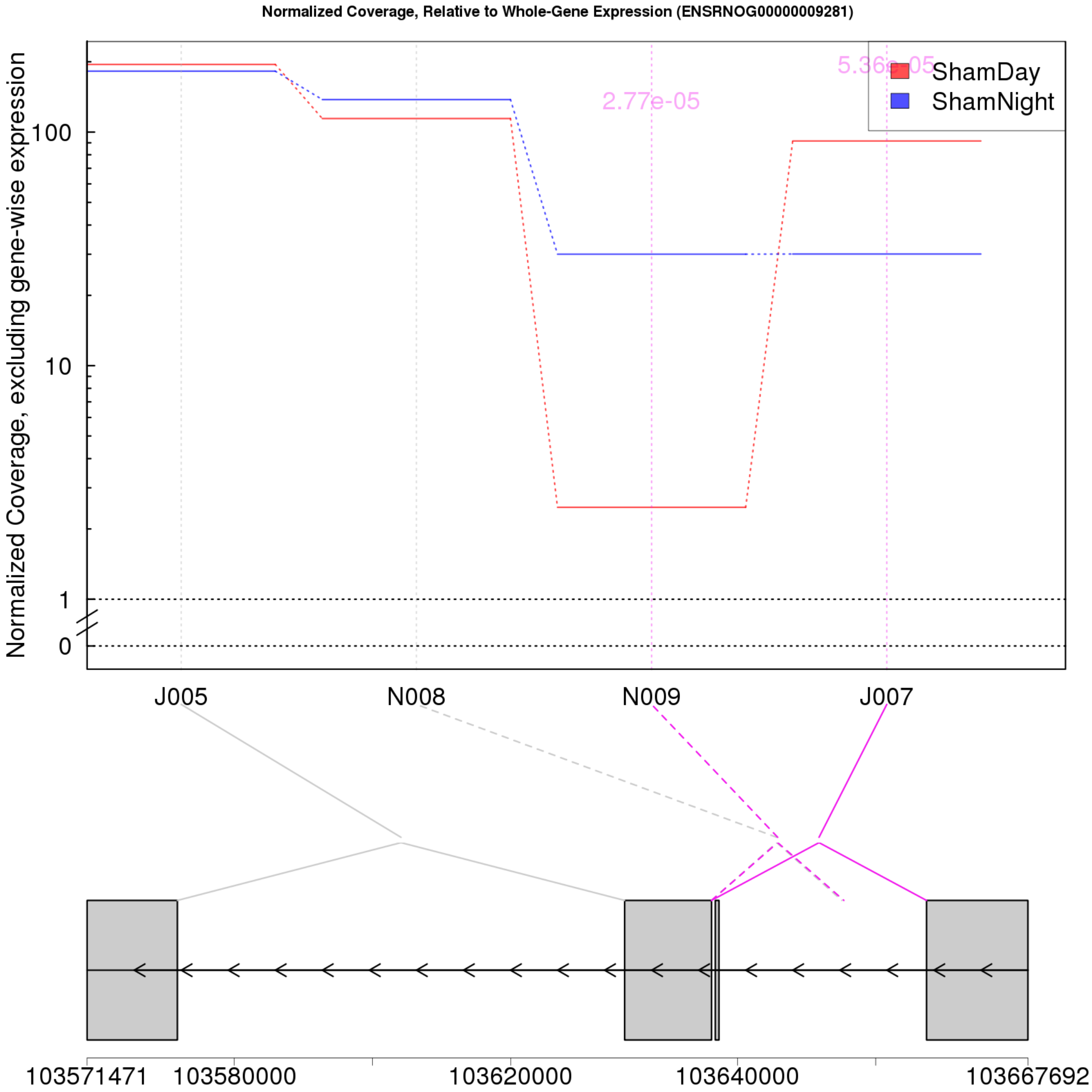

Figure 4: Relative splice junction coverage by biological condition.

Figure 4 displays the "relative" coverage for each splice junction, relative to gene-wide expression. JunctionSeq is designed to detect differential expression even in the presence of gene-wide, multi-transcript differential expression. However it can be difficult to visually assess differential splice junction usage on differentially exprssed genes. This plot displays the coverage relative to the gene-wide expression. These estimates are derived from the GLM parameter estimates (via linear contrasts).

6.2.3 Raw Count Plots

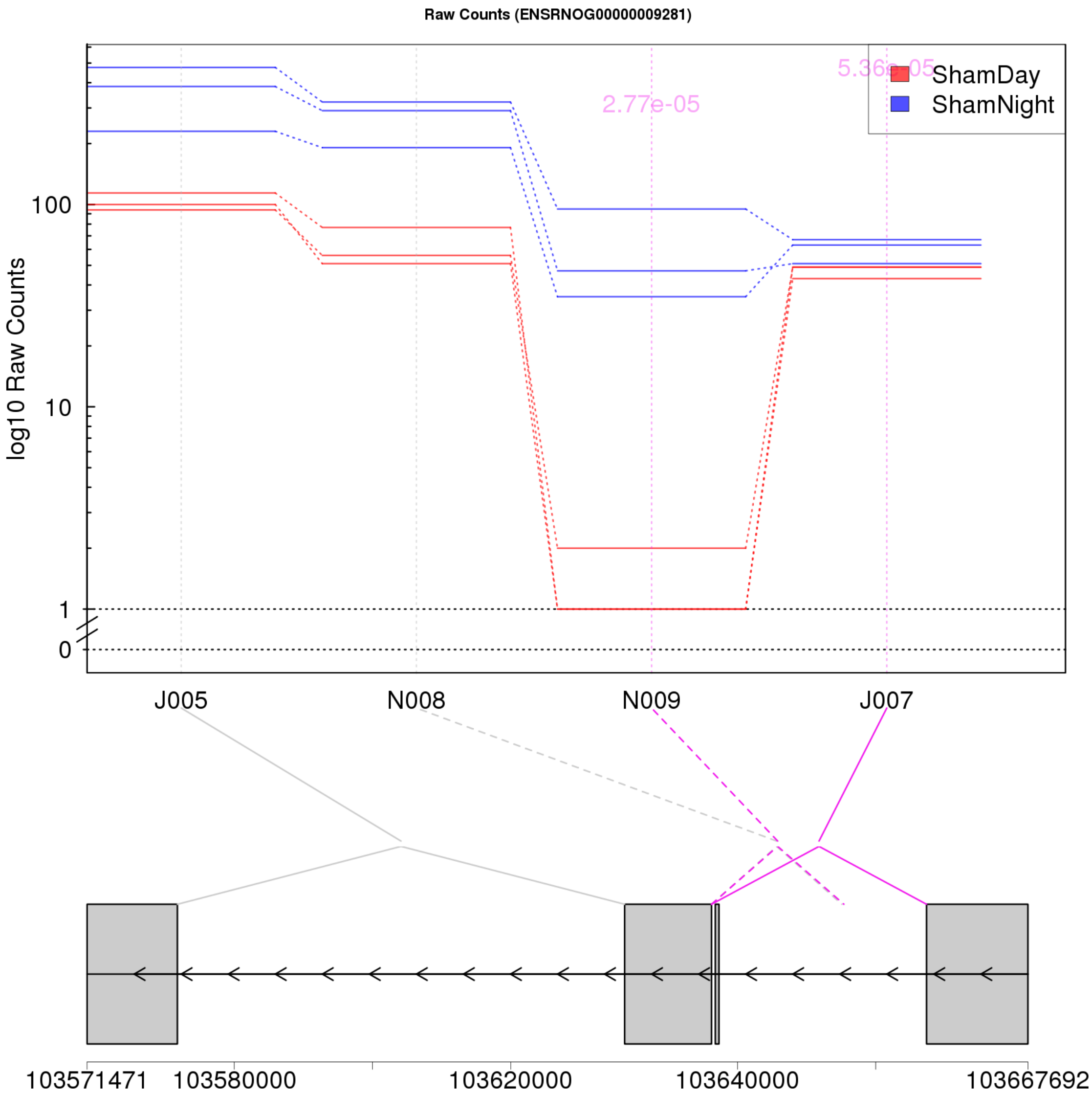

Figure 5: Raw counts for each sample

Figure 5 displays the raw (un-normalized) coverage counts for each sample over each splice junction. This is equal to the number of reads (or read-pairs, for paired-end data) that bridge each junction. Note that these counts are not normalized, and are generally not directly comparable.

6.2.4 Normalized Count Plots

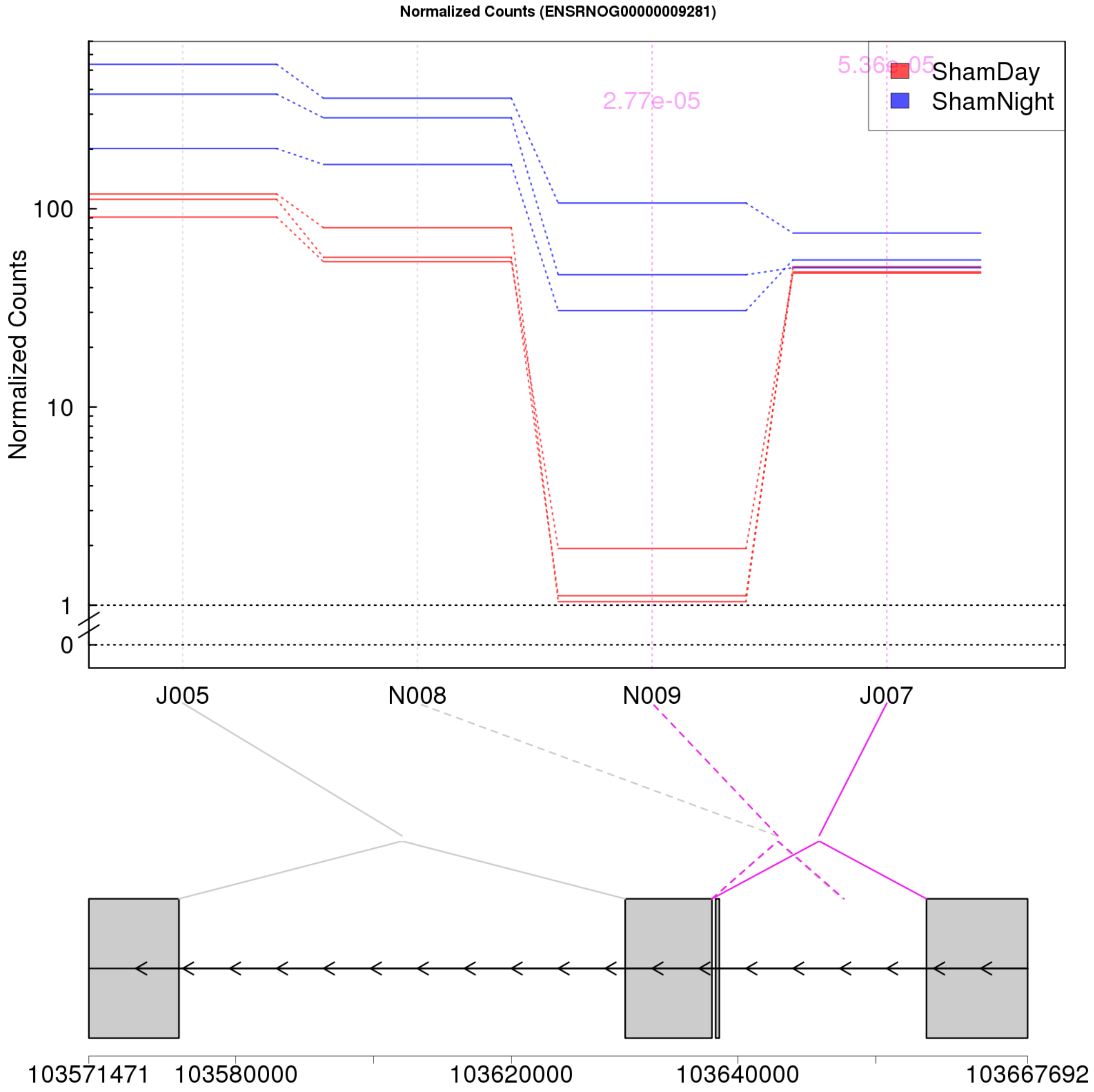

Figure 6: Normalized counts for each sample

Figure 6 displays the normalized coverage counts for each sample over each splice junction. This will be identical to the counts displayed in Section 6.2.3 except that each read count will be normalized by the sample size factors so that the samples can be compared directly.

6.3 Generating Genome Browser Tracks

Once potential genes of interest have been identified via JunctionSeq, it can be helpful to examine these genes manually on a genome browser (such as the UCSC genome browser or the IGV browser). This can assist in a number of ways: it can allow investigators to identify potential sources of artifacts or errors such as repetitive regions or the presence of unannotated overlapping features that may be the actual underlying source of false discoveries. To this end, the QoRTs and JunctionSeq software packages include a number of tools designed to assist in generating simple and powerful browser tracks designed to aid in the interpretation of the data and results.

Advanced tracks like those displayed in Figure **??** can be used to visualize the data and can aid in determining the form of regulatory activity that underlies any apparent differential splice junction usage. The wiggle files needed to produce such tracks can be generated via QoRTs and JunctionSeq. Configuring the multi-colored ”MultiWig” tracks in the UCSC browser require the use of track hubs, the configuration of which is beyond the scope of this manual. More information on track hubs can be found on the UCSC browser documentation.

6.3.1 Wiggle Tracks

Both IGV and the UCSC browser can display ”wiggle” tracks, which can be used to display coverage depth across the genome. QoRTs includes functions for generating these wiggle files for each sample/replicate, merging across technical replicates, and computing mean normalized coverages across multiple samples for each biological condition.

Figure 8 shows an example pair of ”wiggle” files produced by QoRTs for replicate SHAM1_RG1. QoRTs includes two ways to generate such wiggle files from a sam/bam file:

The first way is to create these files at the same time as the read counts. To do this, simply add the ”–chromSizes” parameter like so:

```
java -jar /path/to/jarfile/QoRTs.jar QC \
                   --stranded \
                   --chromSizes inputData/annoFiles/chrom.sizes \
                   inputData/bamFiles/SHAM1_RG1.chr14.bam \
                   inputData/annoFiles/rn4.anno.chr14.gtf.gz \
                   outputData/qortsData/SHAM1_RG1/
```

By default this will cause QoRTs to generate the wiggle file(s) for this sample. Note that if the –runFunctions parameter is being included, you must also include in the function list the function ”makeWiggles”. Note that if the data is stranded (as in the example dataset), then two wiggle files will be generated, one for each strand.

Alternatively, the wiggle file(s) can be generated manually using the command:

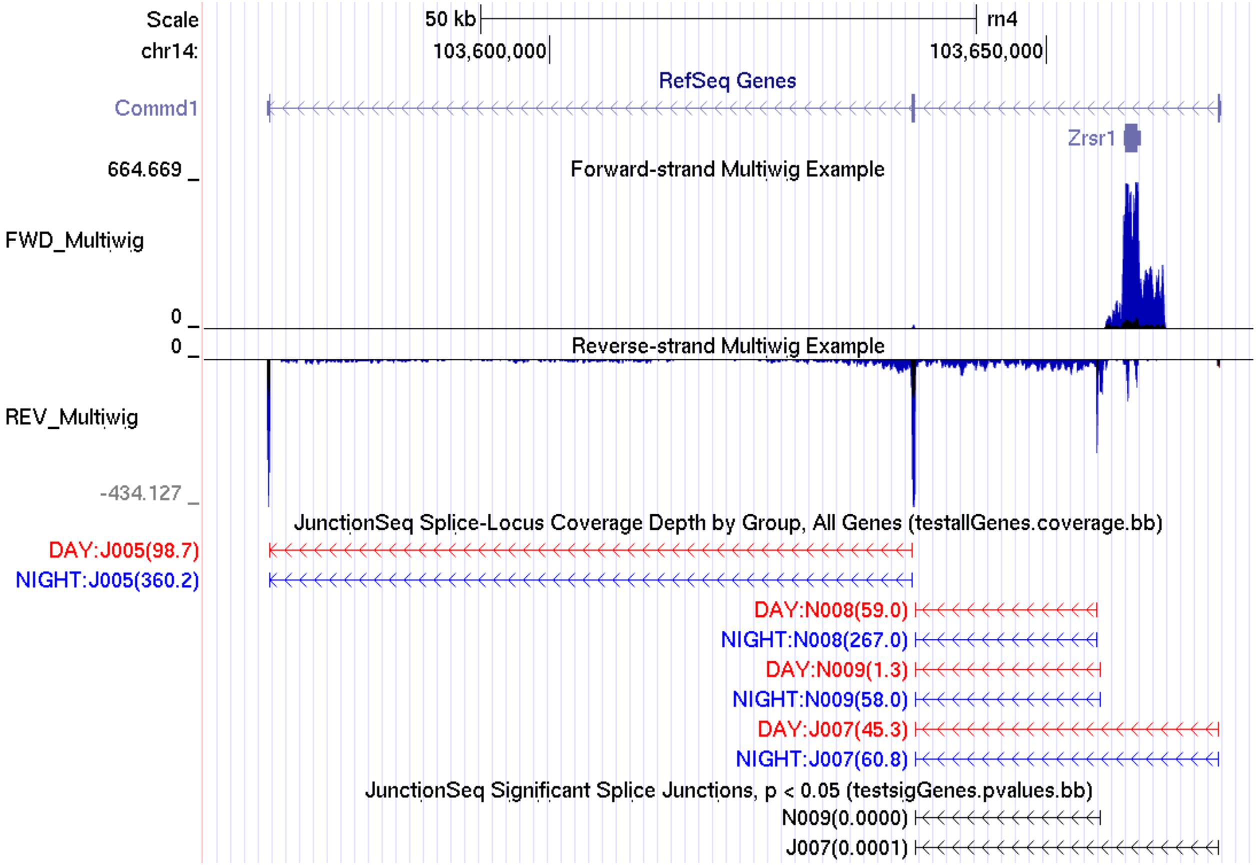

Figure 7: An example of the browser tracks that can be generated using QoRTs and JunctionSeq. The top two "MultiWig" tracks display the mean normalized coverage for 100-base-pair windows across the genome, by biological group (DAY or NIGHT). The top track displays the coverage across the forward (genomic) strand, and the second track displays the coverage across the reverse strand. In both tracks the mean normalized coverage depth across the three NIGHT samples is displayed in blue and the mean normalized coverage depth across the three DAY samples is displayed in red. The overlap is colored black. The third track displays the mean normalized coverage across all testable splice junction loci for each biological condition. Each junction is labelled with the condition ID (DAY or NIGHT), the splice junction ID (J for annotated, N for novel), followed by the mean normalized coverage across that junction and biological condition, in parentheses. Once again, DAY samples are displayed in red and NIGHT samples are displayed in blue. The final bottom track displays the splice junctions that exhibit statistically significant differential usage. Each junction is labelled with the splice junction ID and the p-value. These images were produced by the UCSC genome browser, and a browser session containing these tracks in this configuration is available online here

```
java -jar /path/to/jarfile/QoRTs.jar QC \
                bamToWiggle \
                --stranded \
                --negativeReverseStrand \
```

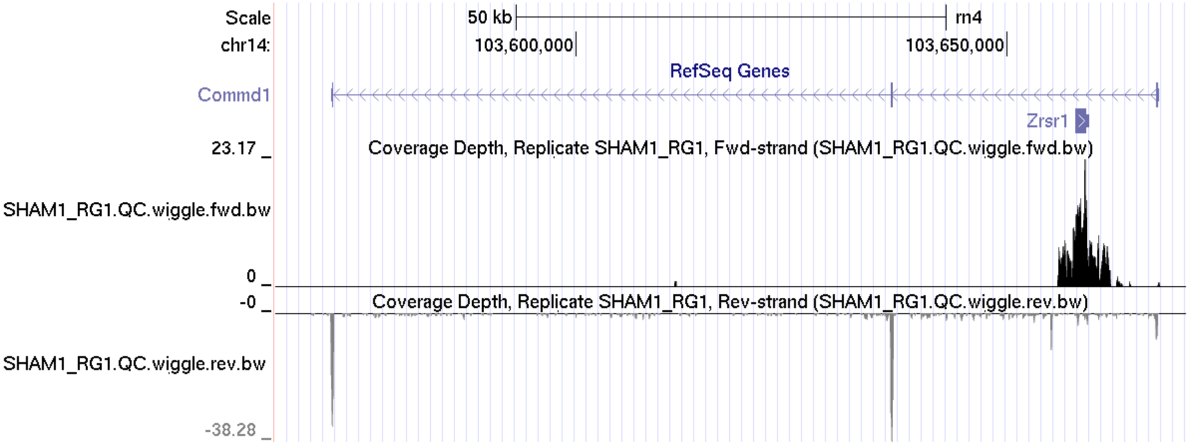
Scale
chr14:
50 kb
rn4
103,600,000
103,650,000
RefSeq Genes
Commd1
Zrsr1
23.17 _
Coverage Depth, Replicate SHAM1_RG1, Fwd-strand (SHAM1_RG1.QC.wiggle.fwd.bw)
SHAM1_RG1.QC.wiggle.fwd.bw
0 _
-0 _
Coverage Depth, Replicate SHAM1_RG1, Rev-strand (SHAM1_RG1.QC.wiggle.rev.bw)
SHAM1_RG1.QC.wiggle.rev.bw
-38.28 _

Figure 8: Two "wiggle" tracks displaying the forward- and reverse-strand coverage for replicate SHAM1_RG1. These tracks display the read-pair mean coverage depth for each 100-base-pair window across the whole genome. The reverse strand is displayed as negative values. These tracks have been loaded into the UCSC genome browser, and a browser session containing these tracks in this configuration is available online here

```
                --includeTrackDefLine \
                inputData/bamFiles/SHAM1_RG1.chr14.bam \
                SHAM1_RG1 \
                inputData/annoFiles/rn4.chr14.chrom.sizes \
                outputData/qortsData/SHAM1_RG1/QC.wiggle
```

If this step is performed prior to merging technical replicates (see Section 4.2), and if the standard filename conventions are followed (as displayed in the examples above), then the technical-replicate wiggle files will automatically be merged along with the other count information in by the QoRTs technical replicate merge utility.

6.3.2 Merging Wiggle Tracks

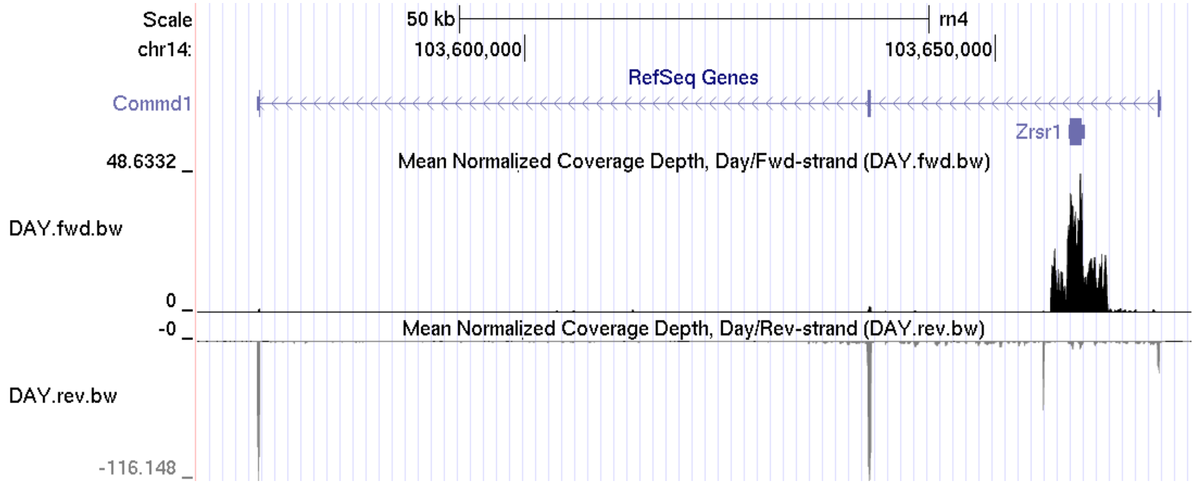
Scale
chr14:
50 kb
rn4
103,600,000
103,650,000
RefSeq Genes
Commd1
Zrsr1
48.6332
Mean Normalized Coverage Depth, Day/Fwd-strand (DAY.fwd.bw)
DAY.fwd.bw
0
-0
Mean Normalized Coverage Depth, Day/Rev-strand (DAY.rev.bw)
DAY.rev.bw
-116.148

Figure 9: Two "wiggle" tracks displaying the forward- and reverse-strand coverage for the three "DAY" samples. These tracks display the mean normalized read-pair coverage depth for each 100-base-pair window across the whole genome. The reverse strand is displayed as negative values. These tracks have been loaded into the UCSC genome browser, and a browser session containing these tracks in this configuration is available online here.

QoRTs can generate wiggle files containing mean normalized coverage counts across a group of samples, as shown in Figure 9. These can be generated using the command:

```
java -jar /path/to/jarfile/QoRTs.jar QC \
               mergeWig  \
               --calcMean \
               --trackTitle DAY_FWD \
               --infilePrefix outputData/countTables/ \
               --infileSuffix /QC.wiggle.fwd.wig.gz \
               --sizeFactorFile sizeFactors.GEO.txt \
               --sampleList SHAM1,SHAM2,SHAM3 \
               outputData/DAY.fwd.wig.gz
```

The "–sampleList" parameter can also accept data from standard input ("-"), or a text file (which must end ".txt") containing a list of sample ID's, one on each line.

6.3.3 Splice Junction Tracks

Splice Junction Tracks are generated automatically by the `writeCompleteResults` function, assuming the `write.bedTracks` parameter is TRUE (the default). By default, three sets of junction tracks are generated:

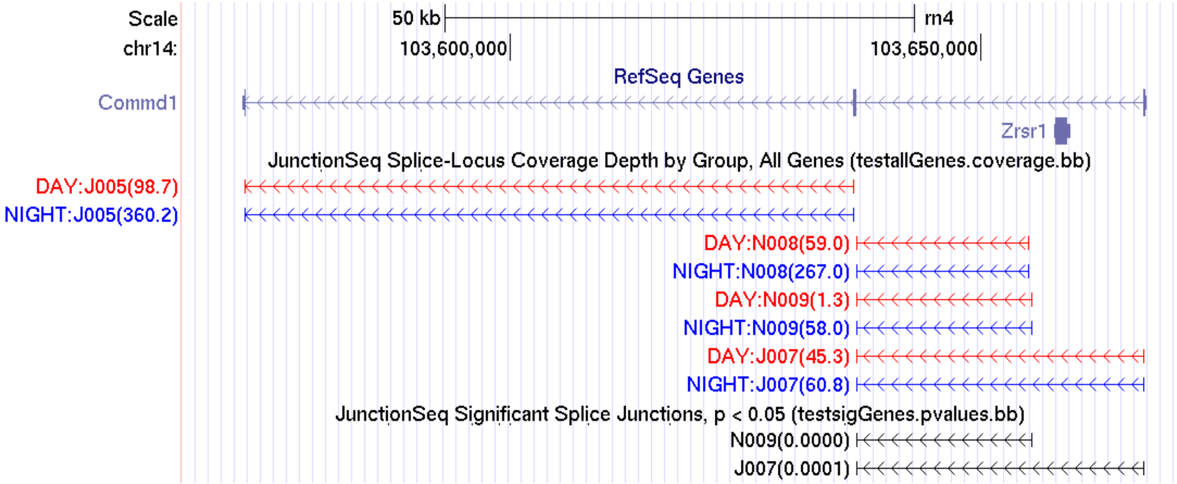

Figure 10: The first track displays the mean-normalized coverage for each splice junction for DAY (blue) and NIGHT (red) conditions. Each splice junction is marked with the condition ID (DAY or NIGHT), followed by the splice junction ID (J for annotated, N for novel), followed by the mean normalized read count in parentheses. The second track displays the splice junctions that display statistically-significant differential usage, along with the junction ID and the adjusted p-value in parentheses (rounded to 4 digits). This track has been loaded into the UCSC genome browser, and a browser session containing these tracks in this configuration is available online here.

- *allGenes.coverage.bed.gz*: A track that lists the mean normalized read (or read-pair) coverage for each splice junction and for each biological condition. These are not equal to the actual mean normalized counts, rather these are derived from the parameter estimates.
- *sigGenes.coverage.bed.gz*: This track is identical to the previous one except that only junctions belonging to genes that contain at least one statistically significant junction are included. All other genes are dropped. This makes the file much smaller and more portable while retaining the most useful information.
- *sigGenes.pvalues.bed.gz*: This track lists all statistically-significant splice junction loci, with the adjusted p-value listed in parentheses.

These three files can be generated manually via the commands:

```
writeExprBedTrack("sigGenes.coverage.bed.gz",
                  jscs = jscs, only.with.sig.gene = FALSE);

writeExprBedTrack("sigGenes.coverage.bed.gz",
                  jscs = jscs);

writeSigBedTrack("sigGenes.pvalues.bed.gz",
                  jscs = jscs);
```

Individual-sample or individual-replicate junction coverage tracks can also be generated via QoRTs. See the QoRTs documentation available online.

7 Statistical Methodology

The statistical methods are based on those described in (S. Anders et al., 2012), as implemented in the DEXSeq Bioconductor package. However these methods have been expanded, adapted, and altered in a number of ways in order to accurately and efficiently test for differential junction usage.

7.1 Preliminary Definitions

For each sample i and each splice junction j we define the junction read (or read-pair) counts:

$$k_{ji}^{\text{Jct}} = \#\text{ reads/pairs bridging junction } j \text{ in sample } i \tag{1a}$$

and

$$k_{gi}^{\text{Gene}} = \#\text{ reads/pairs covering gene } g \text{ in sample } i \tag{1b}$$

The count k_{gi}^{Gene} is calculated using the same methods already in general use for gene-level differential expression analysis (using the "union" rule). Briefly: any reads or read-pairs that cover any part of any of the exons of any one unique gene are counted towards that gene. Reads that cover the exons of multiple annotated genes or that only cover intronic regions are ignored.

7.2 Model Framework

Each splice junction locus is fitted to a separate model. For a given splice junction j located on gene g, we define two "counting bins": $y_1 = (y_{11}, y_{12}, \ldots, y_{1n})$ and $y_0 = (y_{01}, y_{02}, \ldots, y_{0n})$.

Each "counting bin" is a vector of the counts for each sample $i \in \{1, 2, \ldots, n\}$, defined as:

$$y_{1i} = k_{ji}^{\text{Jct}} \tag{2a}$$

and

$$y_{0i} = k_{gi}^{\text{Gene}} - k_{ji}^{\text{Jct}} \tag{2b}$$

Thus, y_{1i} is simply equal to the number of reads spanning the splice junction in sample i, and y_{0i} is equal to the number of reads covering the gene but NOT spanning the splice junction.

Note that while JunctionSeq generally uses methods similar to those used by DEXSeq, this framework differs from that used by DEXSeq on exon counting bins.

In the framework used by DEXSeq, the counting bin count (i.e. k_{gi}^{Gene}) is compared with the sum of all other count bins on the given gene. This means that some reads may be counted more than once if they span multiple features. When reads are relatively short (as was typical when DEXSeq was designed) this effect is minimal, but it becomes less valid as reads become longer.

This flaw is particularly problematic because the dispersion estimation is fitted as a function of the base mean across the model vector. If a large proportion of the reads are counted more than once then this will inflate the base mean counts and may result in an underestimate of the dispersion, resulting in inflated significance. This problem is exacerbated in genes with a large number of features in close succession, and such genes may be disproportionately disposed towards false discovery.

Under our framework, no read-pair is ever counted more than once in each model.

As in DEXSeq, we assume that the count y_{bi} is a realization of a negative-binomial random variable Y_{bi}:

$$Y_{bi} \sim NegBin(\text{mean} = s_i \mu_{bi}, \text{dispersion} = \alpha_j) \tag{3}$$

Where α_j is the dispersion parameter for the current splice junction j, s_i is the normalization size factor for each sample i, and μ_{bi} is the mean for sample i and counting-bin b. Size factors s_i are estimated using the "geometric" normalization method, which is the default method used by DESeq, DESeq2, DEXSeq, and CuffDiff.

7.3 Dispersion Estimation

In many high-throughput sequencing experiments there are too few replicates to directly estimate the locus-specific dispersion term α_j for each splice junction j. This problem is well-characterized, and a number of different solutions have been proposed, the vast majority of which involve sharing information between loci across the genome. JunctionSeq uses the same method used by the DEXSeq package, which is described in detail elsewhere.

Briefly: it has been observed that the dispersion a tends to follow the relation:

$$\alpha(\mu) = \frac{\alpha_1}{\mu} + \alpha_0 \tag{4}$$

The observed locus-specific dispersion estimates $\hat{\alpha}_j$, which are estimated based on the parameter-estimation model (see Section 7.5), are regressed against the model above using a gamma-family GLM. Bins with large residuals are iteratively removed until convergence is achieved. This model fit is used produce parameter estimates $\hat{\alpha}_0$ and $\hat{\alpha}_1$. These are used to produce "fitted" estimates for each splice junction based on the estimated base mean $\hat{\mu}_j$. The ANODEV hypothesis test (see Section 7.4) uses the maximum of the fitted and locus-specific dispersion estimates. This will tend to result in an overestimate of the actual dispersion, which in turn will result in a more conservative hypothesis test.

7.4 Hypothesis Testing

"Differential junction usage" is an observed phenomenon that can arise from numerous forms of junction-specific differential regulation.

Put simply, we are attempting to test whether the fold-change for the biological condition across splice junction j is the same as the fold-change for the biological condition across the gene g as a whole.

This can occur both with and without overall gene-level differential expression. A junction that exhibits "differential usage" might be a junction that exhibits a differential on a gene that is not differentially expressed, or might have constant coverage on a gene that otherwise shows a strong differential. Similarly, a junction could be classified as differentially used if its differential exceeded that of the gene as a whole, or if the differential was in the opposite direction.

In statistical terms: we are attempting to detect "interaction" between the count-bin variable B and the experimental-condition variable C. Thus, two models are fitted to the mean μ_{bi}:

$$H_0 : \qquad \log(\mu_{bi}) = \beta + \beta_b^B + \beta_i^S \tag{5a}$$

$$H_1 : \qquad \log(\mu_{bi}) = \beta + \beta_b^B + \beta_i^S + \beta_{\rho_i b}^{CB} \tag{5b}$$

Where ρ_j is the biological condition (eg case/control status) of sample j.

Note that the bin-condition interaction term ($\beta_{\rho_i b}^{CB}$) is included, but the condition main-effect term ($\beta_{\rho i}^{C}$) is absent. This term can be omitted is because JunctionSeq is not designed to detect or assess gene-level differential expression. Thus there are two components that can be treated as "noise": variation in junction-level expression and variation in gene-level expression. As proposed by Anders et. al., we use a main-effects term for the sample ID (β_i^S), which subsumes the condition main-effect term. This subsumes both the differential and the random variation (noise) in the gene-level expression, improving the power for detecting differential interaction between the count-bin term and the experimental-condition term.

Next, an ANODEV hypothesis test is performed comparing these two models. ANODEV (Analysis of Deviance) is simply a generalization of ANOVA (Analysis of Variance) designed for use on non-normally distributed data, using maximum likelihood rather than ordinary least squares. The ANODEV analysis generates p-values for each splice junction locus, which are then adjusted for multiple testing using the Benjaminiand Hochberg "FDR" method.

These models can easily be extended to include confounding variables: For confounding variable τ, define the value of τ for each sample i as τ_i. Then we can define our null and alternative hypotheses:

$$H_0 : \qquad \log(\mu_{bi}) = \beta + \beta_b^B + \beta_i^S + \beta_{\tau_i b}^{TB} \tag{6a}$$

$$H_1 : \qquad \log(\mu_{bi}) = \beta + \beta_b^B + \beta_i^S + \beta_{\tau_i b}^{TB} + \beta_{\rho_i b}^{CB} \tag{6b}$$

7.5 Estimation

While the described statistical model is robust, efficient, and powerful, it cannot be used to effectively estimate the size of the differential effect or to produce informative parameter estimates.

For the purposes of estimating expression and effect sizes, we create a separate set of generalized linear models. For the purposes of hypothesis testing this model would be less powerful than the model used in section 7.4, but has the advantage of producing more intuitive and interpretable parameter estimates and fitted values. As before, we generate one set of generalized linear models per splice junction locus:

$$H_E : \qquad \log(\mu_{bi}) = \beta + \beta_b^B + \beta_{\rho i}^C + \beta_{\rho_i b}^{CB} \tag{7}$$

Using linear contrasts, the parameter estimates $\hat{\beta}$, $\hat{\beta}_b^B$, $\hat{\beta}_{\rho i}^C$, and $\hat{\beta}_{\rho_i b}^{CB}$ can be used to calculate estimates of the effect size (fold change), as well as the mean normalized coverage over the splice junction for each condition. Additionally, the "relative" expression levels can be calculated for each condition, which indicates the expression of the splice junction, normalized relative to the overall gene-wide expression (which may be differentially expressed).

This model can be extended to include confounding variables in a manner similar to how the hypothesis test models can be extended (as in Equation 6).

$$H_E : \qquad \log(\mu_{bi}) = \beta + \beta_b^B + \beta_{\rho i}^C + \beta_{\tau_i b}^{TB} + \beta_{\rho_i b}^{CB} \tag{8}$$

References

[1] 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, http://arxiv.org/abs/http://bioinformatics.oxfordjournals.org/cgi/reprint/23/21/2881.pdf arXiv:http://bioinformatics.oxfordjournals.org/cgi/reprint/23/21/2881.pdf, doi:10.1093/bioinformatics/btm453.

[2] 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, http://arxiv.org/abs/http://bioinformatics.oxfordjournals.org/content/29/1/15.full.pdf+html arXiv:http://bioinformatics.oxfordjournals.org/content/29/1/15.full.pdf+html, doi:10.1093/bioinformatics/bts635.

[3] 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, http://arxiv.org/abs/http://bioinformatics.oxfordjournals.org/content/26/7/873.full.pdf+html arXiv:http://bioinformatics.oxfordjournals.org/content/26/7/873.full.pdf+html, doi:10.1093/bioinformatics/btq057.

[4] 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.

8 Session Information

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

```
sessionInfo()

## R version 3.1.1 (2014-07-10)
## Platform: x86_64-unknown-linux-gnu (64-bit)
##
## locale:
## [1] C
##
## attached base packages:
## [1] parallel  stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] BiocParallel_1.0.0  JctSeqExData_0.2.01 JunctionSeq_0.3.5
##  [4] Biobase_2.26.0      BiocGenerics_0.12.0 stringr_0.6.2
##  [7] plotrix_3.5-10      statmod_1.4.20      Cairo_1.5-6
## [10] knitr_1.7
##
## loaded via a namespace (and not attached):
##  [1] BBmisc_1.8         BatchJobs_1.5      BiocStyle_1.4.1    DBI_0.3.1
##  [5] KernSmooth_2.23-13 RSQLite_1.0.0      base64enc_0.1-2    brew_1.0-6
##  [9] checkmate_1.5.0    codetools_0.2-9    digest_0.6.4       evaluate_0.5.5
## [13] fail_1.2           foreach_1.4.2      formatR_1.0        highr_0.4
## [17] iterators_1.0.7    sendmailR_1.2-1    tools_3.1.1
```

9 Legal

This software package is licensed under the GNU-GPL v3. A full copy of the GPL v3 can be found in at inst/doc/gpl.v3.txt

This program 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.

This program 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 this program. If not, see http://www.gnu.org/licenses/.

Portions of this software (and this vignette) are "United States Government Work" under the terms of the United States Copyright Act. It was written as part of the authors' official duties for the United States Government and thus those portions cannot be copyrighted. Those portions of this software are freely available to the public for use without a copyright notice. Restrictions cannot be placed on its present or future use.

Although all reasonable efforts have been taken to ensure the accuracy and reliability of the software and data, the National Human Genome Research Institute (NHGRI) and the U.S. Government does not and cannot warrant the performance or results that may be obtained by using this software or data. NHGRI and the U.S. Government disclaims all warranties as to performance, merchantability or fitness for any particular purpose.