JuncBASE

What is JuncBASE?

JuncBASE (**Junc**tion-**B**ased **A**nalysis of **S**plicing **E**vents) is used to identify and classify alternative splicing events from RNA-Seq data. Alternative splicing events are identified from splice junction reads from RNA-Seq read alignments and annotated exon coordinates. In addition to the identification of alternative splicing events, JuncBASE also uses read counts to quantify the relative expression of each isoform and identifies splice events that are significantly differentially expressed across two or more samples. JuncBASE was developed to characterize annotated and novel alternative splicing events throughout *Drosophila* development as well as splice events that are altered upon knockdown of splice factors. It has been further developed and used for alternative splicing analysis in cancer RNA-Seq studies.

How to cite JuncBASE

The method was originally described in:

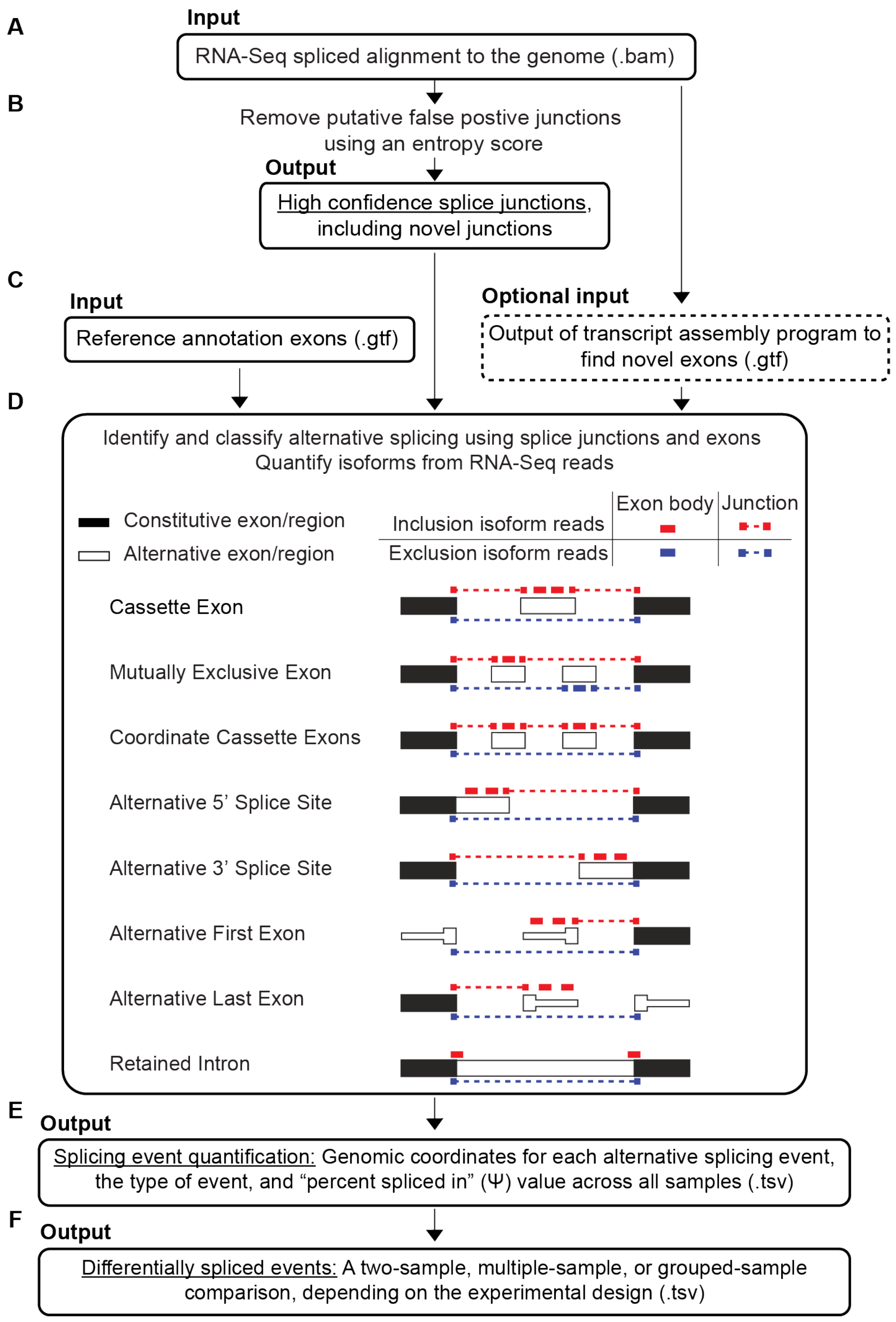
Brooks AN*, Yang L*, Duff MO, Hansen KD, Park JW, Dudoit S, Brenner SE, and Graveley BR. (2011) Conservation of an RNA Regulatory Map between Drosophila and Mammals. Genome Research 21:193-202 *equal contribution link

A standalone manuscript describing the software is currently in progress:

Angela N. Brooks, Kasper D. Hansen, Angadhjot Hundal, Sandrine Dudoit, Matthew Meyerson, Steven E. Brenner. JuncBASE: a junction-based analysis of splicing events from RNA-Seq data

Overview of the JuncBASE analysis pipeline

For each RNA-Seq sample, the spliced alignment is given as input, in BAM format. Splice junctions passing an entropy score threshold are combined with exon coordinates from transcript reference annotations and optional novel transcripts to identify alternative splicing events. The inclusion and exclusion isoforms of each alternative splicing event are quantified using the RNA-Seq read alignments and “percent spliced in” (PSI) values are calculated. Differential splicing analysis is performed from isoform abundances or PSI values, depending on the experimental design.


A
Input
RNA-Seq spliced alignment to the genome (.bam)
B
Remove putative false postive junctions using an entropy score
Output
High confidence splice junctions, including novel junctions
C
Input
Reference annotation exons (.gtf)
Optional input
Output of transcript assembly program to find novel exons (.gtf)
D
Identify and classify alternative splicing using splice junctions and exons
Quantify isoforms from RNA-Seq reads
Constitutive exon/region
Alternative exon/region
Exon body
Junction
Inclusion isoform reads
Exclusion isoform reads
Cassette Exon
Mutually Exclusive Exon
Coordinate Cassette Exons
Alternative 5’ Splice Site
Alternative 3’ Splice Site
Alternative First Exon
Alternative Last Exon
Retained Intron
E
Output
Splicing event quantification: Genomic coordinates for each alternative splicing event, the type of event, and “percent spliced in” (Ψ) value across all samples (.tsv)
F
Output
Differentially spliced events: A two-sample, multiple-sample, or grouped-sample comparison, depending on the experimental design (.tsv)

Requirements

- Python v2.6+ (not Python3)
- Biopython v1.5+: www.biopython.org
- pysam: http://code.google.com/p/pysam/
- R v2.14+ (earlier versions should work): www.r-project.org

- Rpy2: http://rpy.sourcefourge.net/rpy2.html
    - "When compiling R from source, do not forget to specify --enable-R-shlib at the ./configure step" --Rpy2
- MySQL or sqlite (recommended)
    - MySQL, free version v5+(earlier versions should work): www.mysql.com/downloads/mysql/
        - If using MySQL, you will also need:
            - MySQLdb, http://mysql-python.sourceforge.net/MySQLdb.html
    - sqlite, www.sqlite.org
        - pysqlite, http://code.google.com/p/pysqlite/
- Recommended:
    - Bioconductor, limma package. Used for weighted median calculation when creating a virtual reference
    - For plotEntropyScores.py, you will need matplotlib (http://matplotlib.org/) and numpy (http://numpy.scipy.org/).
    - To make plots in compareSampleSets.py, you will need the R package ggplot2: http://ggplot2.org/.

Installation

Download the package of JuncBASE scripts. Unpack (tar -xzf juncBASE_vX.X.tgz). Make sure the juncBASE folder is in your python path.

Quick start and tutorial

Test files

This tutorial will use 6 tissue samples from Illumina Human Body Map 2.0. Reads were aligned using TopHat and alignments from a portion of chromosome 7 were extracted. The BAM files can be found here: http://www.broadinstitute.org/~brooks/juncBASE/test_bams/. To try the tutorial, download all BAMs and the file IlluminaHBM_2.0_samp2test_bam.txt (you may need to modify this file to specify the full path of the bam files).

Other files used:

http://www.broadinstitute.org/~brooks/juncBASE/files/UCSC.hg19.knownGene_w_gene_symbol_EnsemblChr_noGL.gtf

sqlite database of transcript annotations: http://www.broadinstitute.org/~brooks/juncBASE/files/UCSC_hg19_EnsemblChr_noGL

http://www.broadinstitute.org/cancer/cga/tools/rnaseqc/examples/reference/Homo_sapiens_assembly19.fasta.gz (unzip after download)

http://www.broadinstitute.org/~brooks/juncBASE/files/Homo_sapiens_assembly19_introns_noGeneName.txt (Ensembl contains more annotated intron coordinates than UCSC hg19)

Building a *de novo* transcript database

The Cufflinks tools *cufflinks*, *gffread*, and *cuffmerge* should be explicitly specified in the CONSTANTS section of the scripts runCufflinks.py and runCuffmerge.py.

Running cufflinks and cuffmerge takes a long time, so you may skip this step in the tutorial.

```
python juncBASE/runCufflinks.py -i IlluminaHBM_2.0_samp2test_bam.txt --txt_ref UCSC.hg19.knownGene_w_gene_symbol_EnsemblChr.gtf
--out_dir /path/to/cufflinks_denovo/ --num_processes 2
```

```
cd /path/to/cufflinks_denovo/
```

```
find `pwd` | grep transcripts.gtf > cufflinks_denovo_IlluminaHBM2.0.txt
```

```
python juncBASE/runCuffmerge.py -f /path/to/cufflinks_denovo_IlluminaHBM2.0.txt --no_single_exons
-g UCSC.hg19.knownGene_w_gene_symbol_EnsemblChr.gtf -o /path/to/cuffmerge_out/ -s /path/to/Homo_sapiens_assembly19.fasta --tmp_dir
/path/to/tmp_dir
```

There will be an error if the tmp_dir does not already exist.

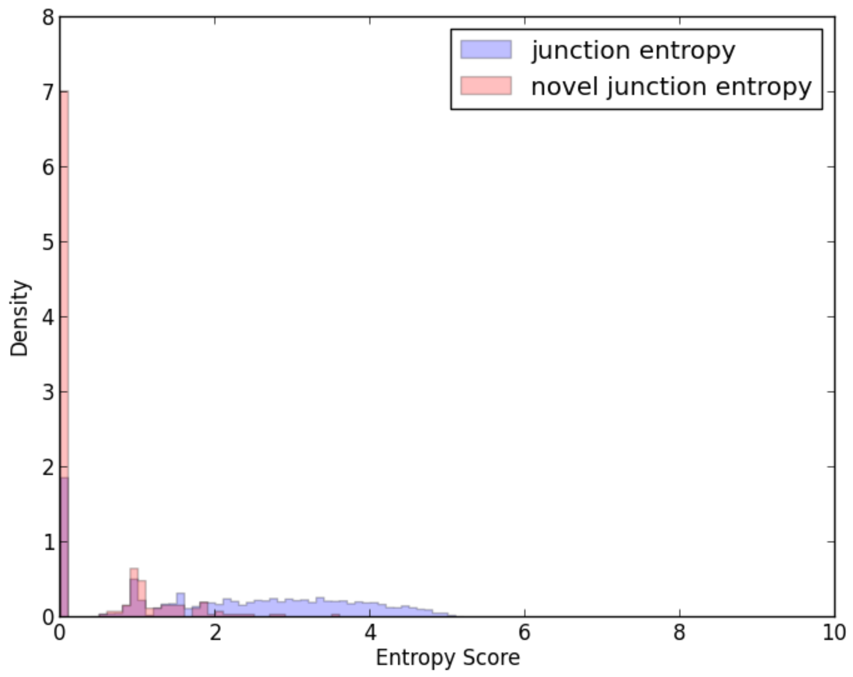
```
python juncBASE/build_DB_FromGTF.py -g /path/to/cuffmerge_out/merged.gtf -d IlluminaHBM_denovo --sqlite_db_dir /path/to/sqlite_db_dir
--initialize
```

```
python juncBASE/build_DB_FromGTF.py -g /path/to/cuffmerge_out/merged.gtf -d IlluminaHBM_denovo --sqlite_db_dir /path/to/sqlite_db_dir
```

Decide on an entropy cutoff to use (adrenal used as an example)

```
python juncBASE/plotEntropyScores.py -s /path/to/adrenal.bam -o adrenal --known_junctions Homo_sapiens_assembly19_introns.txt
```

Resulting .png file:



Since the test bams only contain a portion of a chromosome, this plot will look different than what is expected. You should run this step on all or a subset of samples to see how the entropy distributions differ.

In this entropy plot of annotated and novel junctions, a cutoff of 1 or 1.1 seems reasonable to reduce the number of potential false positive novel junctions.

Step 1) Process SAM/BAM files

(details)

```
python juncBASE/run_preProcess_by_chr_step1.py -i IlluminaHBM_2.0_samp2test_bam.txt -o /path/to/juncBASE_input_files/
--preProcess_options "–unique -j Homo_sapiens_assembly19_introns.txt -c 1.1" -p 2 --nice
```

This step will populate the directory /path/to/juncBASE_input_files/

Processes should normally be checked by using "top", but since these are test input files the processes should complete very quickly. To make sure all processes are complete, the command should be rerun with the added option --check.

Step 1b) Disambiguate splice junction orientations

(details)

```
python juncBASE/disambiguate_junctions.py -i /path/to/juncBASE_input_files/ -g Homo_sapiens_assembly19.fasta --by_chr
```

Step 2) Identify all junctions

(details)

```
python juncBASE/preProcess_getASEventReadCounts_by_chr_step2.py -i /path/to/juncBASE_input_files/ --by_chr
```

Step 3) Create exon-intron junction count files

(details)

```
python juncBASE/run_preProcess_step3_by_chr.py --input_dir /path/to/juncBASE_input_files/ --num_processes 2
```

The processes should complete quickly. Normally, processes would be checked with "top" and by rerunning the command with the option --check.

Step 4) Create a pseudo/"all junction" sample

(details)

```
python juncBASE/createPseudoSample.py -i /path/to/juncBASE_input_files/ -s lung --by_chr
```

Here, "lung" was arbitrarily chosen for the -s option. This can be any of the sample names.

Step 5) Identify alternative splicing events and quantify events from each sample

(details)

These reads are 50bp, so the jcn_seq_len option given is (50-6)*2, with the assumption that confident junction alignments are placed with at least a 6bp overhang on either side of the junction.

```
python juncBASE/run_getASEventReadCounts_multiSample.py -s kidney,breast,adrenal,colon,lung,liver -i /path/to/juncBASE_input_files/ -o /path/to/new/dir/getASEventReadCounts_out --sqlite_db_dir /path/to/sqlite_db_dir --txt_db1 [UCSC_hg19_EnsemblChr_noGL or IlluminaHBM_denovo] --txt_db2 UCSC_hg19_EnsemblChr_noGL --txt_db3 UCSC_hg19_EnsemblChr_noGL --method BH --jcn_seq_len 88 -p 2 --nice --by_chr
```

If you made the *de novo* transcript database, then use IlluminaHBM_denovo for the --txt_db1 option. The processes should complete quickly. Normally, processes would be checked with "top" and by rerunning the command with the option --check.

Step 6) Create tables of raw and length-normalized read counts of exclusion and inclusion isoforms

(details)

```
python juncBASE/run_createAS_CountTables.py -d /path/to/dir/getASEventReadCounts_out/ -i /path/to/juncBASE_input_files/ -s kidney,breast,adrenal,colon,lung,liver --num_processes 2 --jcn_seq_len 88
```

If samples were split by chromosome, check to see that all samples are completed by using top. This step should not take very long (~5-10 min).

```
python juncBASE/combine_createAS_CountTables_by_chr.py -d /path/to/getASEventReadCounts_out/ -o /path/to/new/dir/tutorial_tables
```

Identifying differentially spliced events using the kidney sample as a reference

(details)

```
python juncBASE/pairwise_fishers_test_getASEvents_w_reference.py --in_prefix /path/to/tutorial_tables --jcn_seq_len 88 --output_dir /path/to/new/dir/pairwise_tests/ --out_prefix kidney_comparison --thresh 25 --min_dpsi_threshold 10 --method BH --sign_cutoff 0.05
```

Look in the folder /path/to/new/dir/pairwise_tests/ for the results.

Identifying differentially spliced events using a virtual reference

(details)

Making a virtual reference

```
python juncBASE/run_buildVirtualReference.py --in_prefix /path/to/tutorial_tables
```

Check that no commands from buildVirtualReference.py are running by using top. Then proceed.

Pairwise comparisons with the virtual reference

```
python juncBASE/pairwise_fishers_test_getASEvents_w_reference.py --in_prefix /path/to/tutorial_tables_w_VR --jcn_seq_len 88 --output_dir /path/to/new/dir/pairwise_tests/ --out_prefix vr_comparison --thresh 25 --min_dpsi_threshold 10 --method BH --sign_cutoff 0.05 --has_virtual
```

Look in the folder /path/to/new/dir/pairwise_tests/ for the results.

Differential splicing between two groups of samples

(details)

Note: This approach may not make the most biological sense given the tutorial data, but is given as an illustration of the approach. If two groups of samples are replicates, a t-test may be more appropriate, otherwise Wilcoxon is a better approach.

```
python juncBASE/compareSampleSets.py --in_prefix /path/to/tutorial_tables --all_psi_output /path/to/tutorial_sample_set_comparison.txt
--mt_correction BH --which_test t-test --thresh 10 --delta_thresh 5.0 --sample_set1 lung,colon,breast --sample_set2 kidney,adrenal,liver --html_dir
/path/to/new/example_out_html --pdf --html_out_sign_thresh 1.0
```

The resulting html table from this command will be: /path/to/new/example_out_html/index.html

Reporting PSI values for all alternative splicing (no differential splicing analysis)

(details)

If you just want a table with PSI values for each sample for all identified alternative splicing events, use the --as_only option of compareSampleSets.py

```
python juncBASE/compareSampleSets.py --in_prefix /path/to/tutorial_tables --all_psi_output /path/to/tutorial_allAS.txt --mt_correction BH
--which_test t-test --thresh 10 --delta_thresh 5.0 --sample_set1 lung,colon,breast --sample_set2 kidney,adrenal,liver --as_only
```

All of these options are required, but the only options that are actually used for reporting PSI values are --delta_thresh and --thresh.

Helpful tips

- Each individual sample can be processed through Step 1 autonomously and will not need to be reprocessed again, unless you decide on a different junction entropy cutoff. Step 2 defines the universe of splice junctions based on a set of samples existing in the juncBASE input directory. If the entire analysis pipeline is run on a set of, say, 10 samples and a week later 5 new samples are completed, the first 10 samples will not have to go through Step 1 again, but all steps after Step 1 will need to be re-run with the additional 5 samples. Adding more samples may lead to more splice junctions that did not exist in the previous set.

- Multiprocessing tips. Most of the multiprocessing runs are actually just wrapper scripts that run parallel system calls. It is recommended that jobs are monitored using *top.* There are also helpful options such as --force, and --check, that will either force rerunning all jobs or print out messages as to which runs are not completed yet. By default, these will avoid rerunning jobs that already have completed output files (i.e. it performs job avoidance). The wrapper scripts provide convenience, but may not work well with all systems.
    - Troubleshooting tips when having problems with the multiprocessing steps:
        - Make sure the "python" command is referencing the version of python you want to use on your system.
        - The default shell is tcsh. If using a different shell, change the SHELL variable at the top of the script. To check what shell you are using, you can run: set | grep shell

- What if my reads are of different lengths? It is not recommended that splicing analysis be performed on reads of different length as the length of the reads affects the ability to align to splice junctions. You should still be able to run JuncBASE with reads of different length, but it is recommended to trim reads to make them all the same length and realigning the reads prior to running JuncBASE or any other analysis.

Manual: Identifying and quantifying alternative splicing

For each python script, you can view all options by simply typing:
python <script.py>

Make sure that all scripts and packages are in your PYTHONPATH and all are in the same directory. If you receive errors that files cannot be opened, check the CONSTANTS section from each script to make sure that the paths are correctly specified.

Some steps in the pipeline include a wrapper script that makes it easy to run the step on each sample and on each chromosome individually. The multiprocessing capability is useful if a compute cluster is accessible. The wrapper script will run commands to the main script using each sample and chromosome combination.

What follows is a more detailed explanation of the steps of the JuncBASE analysis pipeline. **For a tutorial and quick start, please go here.**

Build an annotation database.

The annotation database is used to identify all exons in the transcriptome and it is derived from a GTF file. It is recommended that you create two different databases. The first will be used to identify all internal exons (not the first or last exon in a transcript). I would suggest using a GTF that combines your Cufflinks identified transcripts with those that are annotated in a reference set (see instructions). The second database should be derived from just a reference annotation. This second database is used to define alternative first and last exons, *de novo* transcripts are typically not good for this since there are a lot of truncated transcripts; however, if you trust the *de novo* transcript annotations, then you only need to generate one database.

In addition, the GTF file should have "transcript_id"s grouped into "gene_id"s. For example, Ensembl GTFs tend to be in the correct format. GTF files downloaded from the Table Browser from UCSC use the same ID for both transcripts and genes. A specification of the GTF format can be found here: http://mblab.wustl.edu/GTF22.html. It is important to have "transcript_id"s grouped into "gene_id"s for the identification of alternative first and last exons. The pipeline will still run if transcripts are not grouped in this way, but without the report of the alternative first and last exons.

If using Ensembl annotations, use only GTF lines that have "exon" in the third column. Rows with "gene" in the third column give the same identifier in the "transcript_id" and "gene_id" field, which causes errors when building the database.

For each annotation database you will need to run the script (if using mysql):

```
python build_DB_FromGTF_mysql.py -g <GTF file> -d <Database name>
```

or (if using sqlite):

```
python build_DB_FromGTF.py -g <GTF file> -d <Database name> --sqlite_db_dir
<Directory to put sqlite database files> [--initialize]
```

For build_DB_FromGTF.py, you run the command twice. The first time with --initialize (this sets up the tables in the database) then again without --initialize.

Remember the database names you use here because you will use them later.

After the script is finished building, please check that the database contains entries in the gene, exon, and intron tables. I suggest looking at a few rows from these tables (e.g., SELECT * FROM exon LIMIT 5) and also doing a count of the entries to make sure this is around the number you expect from the gtf file (e.g., SELECT COUNT(*) FROM gene).

Optional) Build a *de novo* transcript database.

JuncBASE identifies and classifies alternative splicing events from exon coordinates specified in a GTF file and from spliced junction alignments directly from the RNA-Seq reads. JuncBASE can also use exon coordinates from a *de novo* transcript assembly program such as Cufflinks (http://cufflinks.cbcb.umd.edu/). Cufflinks has been used for testing; however, any transcript assembly program can be used as long as the transcripts are given in GTF format. *Make sure that the GTF file of de novo transcripts also contains annotated transcripts*. Once a GTF file has been made, use build_DB_FromGTF.py to build an additional database of the *de novo* transcripts with annotated transcripts.

Scripts are included to assist in transcript assembly using Cufflinks. The Cufflinks tools *cufflinks*, *gffread*, and *cuffmerge* should be explicitly specified in the CONSTANTS section of the scripts runCufflinks.py and runCuffmerge.py. Please review the options used for the *cufflnks*, *gffread*, and *cuffmerge* commands in runCufflinks.py and runCuffmerge.py to ensure that these fit with your input data.

Step 0) Check SAM/BAM files.

It is assumed that you have already aligned your reads using something like TopHat and that your alignments are in SAM/BAM format. It is highly recommended that there is a tag for junction reads that is "XS:A:+" or "XS:A:-" which gives the strand of the junction read. This is default by TopHat. The XS tag is not required for the BAM files. If they are not present, JuncBASE will infer the strand of the junction alignment based on the corresponding splice site sequence (e.g., if the corresponding intron starts with "GT" and ends with "AG", it is assumed to be on the "+" strand.)

Take special note of the format of the chromosome names used in the SAM/BAM file and the GTF annotation files. For easy usage with the UCSC Genome Browser, JuncBASE converts chromosome format to use "chr". Human and *Drosophila melanogaster* genomes should work without modifications; however, there may be issues parsing information from other genomes with non-traditional chromosome names.

Step 0, optional) Determine junction entropy cutoff.

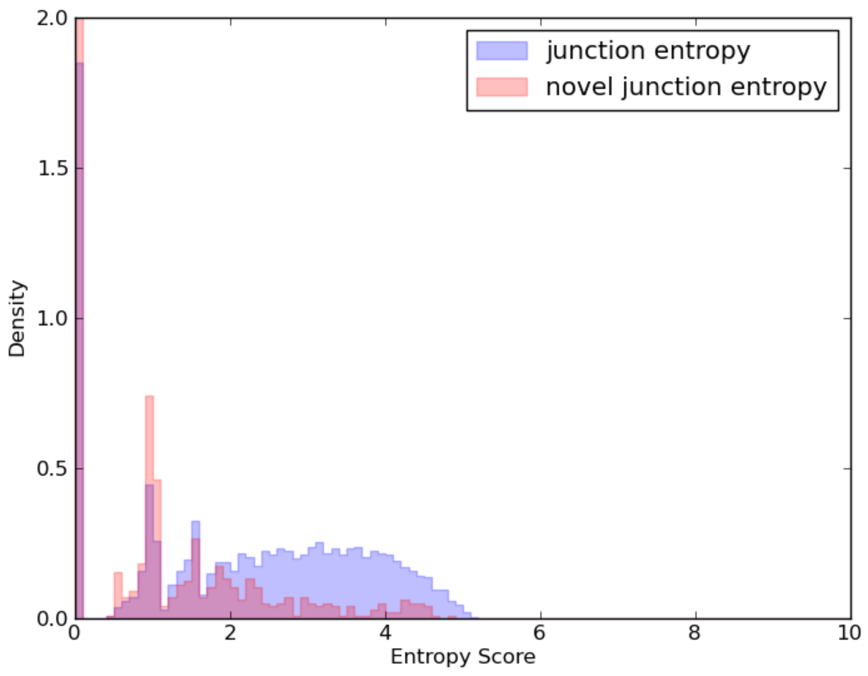
The script, plotEntropyScores.py, can be used to identify what entropy score cutoff to use for your samples. Running plotEntropyScores.py will result in a .png file of a histogram of entropy scores for all exon-exon junctions in the spliced alignment. It is recommended to include a list of annotated intron coordinates so that annotated and novel exon-exon junction entropy scores can be plotted together. Looking at the distribution of entropy scores between annotated and novel junctions helps in identifying a score cutoff that removes potential false positive novel junctions with low entropy scores.

```
python plotEntropyScores.py
```

-s <BAM/SAM file>	

-o <sample name>	The name will be used for all output files. e.g., untreated
--known_junctions <file>	Optional: Intron coordinates give in tab-delimited format: <chr>\t<start>\t<end>\t<strand>. The start and end positions are 1-based coordinates.

An example histogram is here:



Step 1) Process SAM/BAM files.

In these instructions, I'm assuming you have two or more RNA-Seq samples for which you are trying to compare splicing. We will call them Untreated and Treated for these examples, but any name can be used for the sample name. Run each SAM file separately through the script preProcess_getASEventReadCounts.py.

```
python preProcess_getASEventReadCounts.py
```

-s <SAM/BAM file>	
-n <sample name>	Name will be used as a prefix for all output files, e.g., untreated, treated
-c <float>	The minimum entropy score for a junction to be kept for further analysis. Default=1.0. You can use plotEntropyScores.py to help decide on an appropriate cutoff.
-o <dir>	Directory for JuncBASE input_files used in later steps
-j <file>	Optional: Intron coordinates give in tab-delimited format: <chr>\t<start>\t<end>\t<strand>. The start and end positions are 1-based coordinates.

The -c option filters junctions based on an entropy score. You can just use the default value by not including this option; however, you should use the results from the plotEntropyScores.py to get a sense of the entropy score distribution your reads. The right cutoff will depend on the library preparation, the depth of sequence, and the diversity of transcripts present in your samples. The -j option specifies splice junctions (given as intron coordinates) that should be kept regardless of the entropy score and is usually used for annotated splice junctions.

From this step, for each sample, you should get a .bed file which contains all junctions and a file called "..._genome_reads.txt.gz" which is a flat text file that represents all the non-junction reads from the SAM/BAM file. **The score field in the BED file gives the number of reads that support the junction and can be directly uploaded to the UCSC Genome Browser for visualization.**

It is VERY IMPORTANT to use the recommended directory structure.
e.g.,
input_files/untreated
input_files/treated

or

input_files/[samp1_name]
input_files/[samp2_name]
input_files/[samp3_name]
...
input_files/[sampn_name]

I would recommend that the subdirectories of input_files be the same name given in the -n option for preProcess_getASEventReadCounts.py.

Multiprocessing version

For this multiprocessing version, you will need a tab-delimited file (no header) associating the sample name (column 1) with the path of the sam/bam file (column 2). If you use the multiprocessing version of the JunBASE pipeline in the beginning, you must use this option in all subsequent steps.

```
python run_preProcess_by_chr_step1.py [options] [--nice, --check, --force]
```

-i <tsv file>	Tab-delimited file associating the sample to a BAM file. <sample name>\t<bam file location>
-o <dir>	Directory for JuncBASE input_files
--preProcess_options	Options passed to preProcess_getASEventReadCounts.py, given in "". e.g., "–unique -j /path/to/intron/coordinates"
-p <int>	Number of simultaneous runs

Running this step with the tsv file given in -i and specifying a root directory will automatically create the appropriate directory structure. For example, if the samp2bam file contained:

```
untreated /path/to/untreated.bam
treated /path/to/treated.bam
```

and the -o option was given as /path/to/input_files, the subdirectories would be automatically created as follows:

/path/to/input_files/untreated/untreated_chr1/...
/path/to/input_files/untreated/untreated_chr2/...
/path/to/input_files/untreated/untreated_chr3/...

...

/path/to/input_files/treated/treated_chr1/...
/path/to/input_files/treated/treated_chr2/...
/path/to/input_files/treated/treated_chr3/...

and so forth.
Before proceeding to the next step, run the script run_preProcess_by_chr_step1.py again, but with the --check option to make sure that all files were created properly. By default, the script will avoid re-running completed jobs; however, all jobs can be forced to run with the --force option.

Step 1b, optional) Disambiguate strand of splice junctions

Occasionally, TopHat will report the same splice junction as having the opposite strand orientation in different samples, which may cause errors in later steps. This script will use the genome reference sequence to infer the true orientation of splice junctions that are reported to have different strand orientations.

```
python disambiguate_junctions.py -i <root directory of JuncBASE input files> -g
<genome sequences in .fasta format> [--by_chr, if multiprocessing]
```

Step 2) Identify all junctions.

This step will create one BED file that combines junctions from all samples.

```
python preProcess_getASEventReadCounts_by_chr_step2.py -i <root directory of
JuncBASE input files>
```

The BED file will be located in /path/to/input_dir/tmp_preProcess_getASEventReadCounts_step2.bed. Were the /path/to/input_dir is given as the -i option.

Multiprocessing version

```
python preProcess_getASEventReadCounts_by_chr_step2.py -i <root directory of
JuncBASE input files> --by_chr
```

The BED files created in the input_dir given in -i option and will be named tmp_[chr name]_preProcess_getASEventReadCounts_step2.bed

Step 3) Create exon-intron junction count files

This step will be run separately for each sample.

```
python preProcess_getASEventReadCounts_step3.py
```

-i <dir>	Directory of JuncBASE input files
-n <sample name>	e.g., untreated or treated
-t <file>	Exact location of the the file created in Step2. i.e., /path/to/input_files/ tmp_preProcess_getASEventReadCounts_step2.bed
--min_overhang <int>	This is the minimum length of the alignment on either side of the exon-intron boundary to confidently assign the read to this exon-intron junction. The recommended value is 6.

This step will take some time to run. It will create another file called <sample name>_intron_exon_junction_counts.txt, depending on the -n option. This file gives counts of the number of reads aligning to every exon-intron junction. This file is used for quantifying intron retention, alternative 5' splice sites, and alternative 3' splice sites.

Multiprocessing version

```
python run_preProcess_step3_by_chr.py [--force --check]
```

-input_dir <dir>	Directory of JuncBASE input files
--num_processes <int>	Number of simultaneous runs

Step 4) Create a pseudo/"all junction" sample

The pseudo sample is only used to define the set of all junctions and is not used for quantification. The pseudo sample will link to the BED file(s) created in Step 2. An arbitrary sample is given in order to create other pseudo files that are actually not used.

```
python createPseudoSample.py [options] [--by_chr]
```

-i <dir>	Directory of JuncBASE input files
-s <sample_name>	An arbitrary sample name (e.g., untreated)

If samples have been run with the multiprocessing version, then --by_chr will need to be added to the command.

Step 5) Identify alternative splicing events and quantify events from each sample

This is the major step of JuncBASE that identifies, classifies, and quantifies alternative splicing events. This step runs system calls individually on each sample and each chromosome if specified.

```
python run_getASEventReadCounts_multiSample.py [options] [--by_chr, if
multiprocessing]
```

-s <comma-separated string or file>	Comma separated list of samples to run or a file with the names of the samples to run.
-i <dir>	Directory of JuncBASE input files
-o <dir>	Root directory for a place to put all output. **NOTE:** This should be different than the -i option.
--txt_db1 <database_name>	Name of the database with all exon and intron annotations. It is recommended that this is the *de novo* transcript database created in Step 0. It should be the same names used in the -d option of build_DB_FromGTF.py.
--txt_db2 <database_name>	Name of the database to define first and last exons. It is recommended that you use a confident transcript annotation database; however, the same *de novo* transcript database can be used if the first and last exons in the annotation are trusted.
--txt_db3 <database_name>	Optional, but recommended: Database of transcript annotations derived from a gtf file. Used for annotating gene names and whether an intron/junction is annotated or not. This should be a standard reference annotation. By default, txt_db1 is used for this information.
--jcn_seq_len <int>	I recommend this value to be (read length -6)*2 which assumes reads aligned to junctions with at least a 6bp overhang. If TopHat was used for the alignment and you used the -a option with something < 6, give the value (read length - anchor length)*2
-p <int>	Number of processes to run simultaneously. Def=2
--nice	Optional: Will run jobs locally using nice
--force	Optional: By default, will check for the existence of the final output of all runs. This option will force to rerun everything.
--check	Will check samples that are done and print out which still needs to be run
--print_cmd	Will print commands that will be run, but will not run them. This is useful for debugging.

Specify database:

If using sqlite

--sqlite_db_dir <dir>	Location of sqlite databases. This should be the same as was used in build_DB_FromGTF.py

If using MySQL

--host <host name>	MySQL database host. Def='localhost'
--user <user name>	MySQL database user. Def='root'
--passwd <passwd>	MySQL database password. Def=''

If input files were split by chromosome, then add "–by_chr" option.

The result of this step is a table with details of each alternative splicing event identified in the set of samples examined. If the analysis was performed by chromosome, this information is located in files named /path/to/out/dir/[samp]/[samp]_chr[X]/[samp]_chr[X]_all_AS_event_info.txt, where the /path/to/out/dir is specified by the -o option. The details of each splicing event include coordinates, and raw and length-normalized read counts to specific junctions and regions involved in the alternative splicing event. These details can be useful for custom analyses, but most users will not look at these intermediate files. Columns are explained in a file that comes with the JuncBASE package: README_getASEventReadCounts_intermediate_outputs.txt.

There are additional intermediate files that are produced, but deleted by default. To keep these intermediate files, use the --keep_intermediate option.

Step 6) Create tables of raw and length-normalized read counts of exclusion and inclusion isoforms

This step collects information from runs performed in Step 5 and reports the AS event and read counts for every event in every sample in a table.

```
python createAS_CountTables.py [options]
```

-d <root_dir>	Output directory from Step 5 (the -o option from Step 5)
-o <output_file_prefix>	Prefix of output files. Will create output to the current directory unless a full path is given in the prefix. e.g., -o my_data (new files in the current directory) -o /path/to/dir/to/put/files/my_data (will put output in /path/to/dir/to/put/files/)
--jcn_seq_len <int>	Same value used in Step 5
-s <samples>	Comma-separated list of the samples or a file of the sample names. The order in which they are given is the order in which they will appear in the columns of the output table. If one of the samples is a reference (e.g., Control or Untreated), this sample should be listed first.

Multiprocessing version

This is a two step process if analysis was split by chromosome. The first step creates temporary files of counts, organized by chromosome and the second step merges all chromosomes into one file.

```
python run_createAS_CountTables.py [options]
```

-d <root_dir>	Output directory from Step 5 (the -o option from Step 5)

-i <input_dir>	Directory of JuncBASE input files (the -i option from Step 5)
--jcn_seq_len <int>	Same value used in Step 5
-s <samples>	Comma-separated list of the samples or a file of the sample names. The order which they are given is the order in which they will appear in the columns of the output table. If one of the samples is a reference (e.g., Control or Untreated), this sample should be listed first.
--num_processes <int>	Number of processes to run simultaneously. DEF=2

To combine all runs, first make sure that all processes are done from run_createAS_CountTables.py first by using "top":

```
python combine_createAS_CountTables_by_chr.py [options]
```

-d <root_dir>	The same directory specified in run_createAS_CountTables.py
-o <file_prefix>	Prefix of output files. Will create output to the current directory unless a full path is given in the prefix. e.g., -o my_data (new files in the current directory) -o /path/to/dir/to/put/files/my_data (will put output in /path/to/dir/to/put/files/)
--remove_tmp_files	Optional: Will remove the temporary files created in the run_createAS_CountTables.py step.

Output file explanation

Six final output files will be created from this step. All files will start with the prefix specified in -o. First 11 columns are described in the "Percent Spliced In" table section (here). The *_counts.txt tables give the raw read counts to the exclusion and inclusion isoforms of each event. The format of the counts are "exclusion;inclusion". The *_counts_lenNorm.txt" give length normalized counts of the exclusion and inclusion isoforms, where the raw read counts are divided by per 100bp of the exon model of each isoform.

Manual: Differential splicing analyses

Pairwise Fisher's Exact test with a reference sample

If one sample is a reference sample, then all other samples can be compared against it to identify differentially spliced events. Fisher's Exact tests are performed on raw read counts from 2x2 tables of exclusion and inclusion read counts between the reference and each sample. The reference sample has to be listed first in the -s option of Step 6 (createAS_CountTables.py). This approach performs pairwise comparisons between the reference and all samples for each splicing event; therefore, this approach will be slow if your sample size is large (e.g. hundreds).

```
python pairwise_fishers_test_getASEvents_w_reference.py [options]
```

--in_prefix <file_prefix>	Prefix of output files created from createAS_CountTables. In createAS_CountTables, this is the -o option
--jcn_seq_len <int>	Same value used in Step 5
--output_dir <dir>	Directory to put the output files to this step
--out_prefix <string>	Prefix added to all output files. Suggested values include the name of the analysis you are doing and a date. Def=None
--thresh <int>	Threshold for the minimum number of total reads in an event to consider it expressed enough for testing. Def=25

--min_dpsi_threshold <float>	Threshold for the minimum difference in Percent Spliced In (PSI) values between the sample with the smallest and largest PSI. Events with delta PSI values below the threshold will not be tested or reported. Def=5.0
--method <BH, bonferroni>	Multiple testing correction method: "BH" - Benjamini & Hochberg,"bonferroni". Must select these strings as the option. BH is recommended.
--sign_cutoff <float>	Cutoff of corrected p-value significance. Default=0.05

Output files

Look in the directory specified by --output_dir for the results.

File suffix	Description
*_allPSI.txt	A table of all PSI values for all alternative splicing events that passed the expression threshold. "NA" values mean that in the specific sample, the splicing event was not expressed. The first 11 columns follow the standard PSI table output.
*_sign_by_samp_PSI.txt	A table of PSI values for alternative splicing events that were significantly different from the reference sample. Samples not passing expression thresholds or the significance cutoff are indicated with "NA". Here, the multiple-testing correction is across the multiple splicing events tested. The first 11 columns follow the standard PSI table output.
*_most_sign_PSI.txt	A table of PSI values for alternative splicing events that were significantly different from the reference sample. Samples not passing expression thresholds or the significance cutoff are indicated with "NA". Here, the multiple-testing correction is across the multiple splicing events tested AND all samples tested. The first 11 columns follow the standard PSI table output.
*_pairs_p_val.txt	Most users will not need to use this table: This table gives the raw p-values and corrected p-values for each event and each pairwise comparison. The first 11 columns follow the standard PSI table output. The next two columns indicate the sample indices that were compared. The last 3 columns give: raw p-value, sign_by_samp corrected p-value, most_sign corrected p-value.
*_left_intron_retention_allPSI.txt	Most users will not need to use this table: PSI values for the left (5' most splice site relative to the genome) side of each intron retention event.
*_right_intron_retention_allPSI.txt	Most users will not need to use this table: PSI values for the right (3' most splice site relative to the genome) side of each intron retention event.

Pairwise Fisher's Exact test with a virtual reference sample

Creating a virtual reference

A virtual reference can be created for each analysis. A virtual reference is created for each splicing event by taking the median total isoform abundance of the event and the median PSI values for the event and creates virtual read counts.

```
python run_buildVirtualReference.py [options]
```

--in_prefix <prefix string>	Prefix of output files created from createAS_CountTables. In createAS_CountTables this was the -o option

--total_thresh <int>	Threshold for the minimum number of total reads in an event to consider it expressed enough for testing. Default=25
--remove_from_median <string>	Optional: Comma separated list of samples that will not be used when calculating the median.
--weights <string>	Comma separated list of weights given in the order of the samples in the table. Weights are used to create a weighted median. Default is equal weight for all samples. The order of the weights correspond to the order which the samples appear in the table.
-p <int>	Number of processes to run simultaneously. Default=2

Check that all runs are completed with top.

There will now be additional files that contain exclusion and inclusion counts for each splicing event for each sample, but will have an additional column for a virtual reference. These files will have a prefix that is reported after running this step.

e.g., Output files were created with the following prefix: [in_prefix]_w_VR

Pairwise comparison with the virtual reference

The same script for pairwise comparison with a given reference is used for comparison with a virtual reference: pairwise_fishers_test_getASEvents_w_reference.py. The --in_prefix options should refer to the tables created from run_buildVirtualReference.py: [in_prefix]_w_VR. You must run pairwise_fishers_test_getASEvents_w_reference.py with the additional option --has_virtual. All other options are the same.

Differential splicing between two groups of samples

This analysis will identify differentially spliced events between two specified groups of samples. The PSI values are used for the comparison. The resulting output table will be the standard PSI table, with additional columns. As an option, an html table will be created which links differentially spliced events with box plots for helpful visualization. This script can also be used for any generic table using the -i and --generic options, specifying the 0-based index of the first column containing values.

```
python compareSampleSets.py [options]
```

--in_prefix <prefix>	Prefix of output files created from createAS_CountTables. In createAS_CountTables this was the -o option.
--all_psi_output <outfile>	Output file that will contain the PSI values for all events and samples. The last two columns will correspond to the raw p-value and corrected p-value. If a generic file is used, this will be the output file
--thresh <int>	Threshold for the minimum total abundance in an event. Default=10
--mt_correction <BH or bonferroni>	Multiple testing correction Method: "BH" - Benjamini & Hochberg, "bonferroni". Must select these strings as the option.
--which_test <t-test or Wilcoxon>	Which test to use. Either "t-test" or "Wilcoxon". Default=Wilcoxon
--delta_thresh <float>	Minimum PSI (or generic value) difference between the maximum and minimum values for a given event to be considered a change. Default=5.0
--sample_set1 <string or file>	Comma delimited list of samples in set 1 or a file with a list of names, one per line. Names must be in header columns of input files.
--sample_set2 <string or file>	Comma delimited list of samples in set 2 or a file with a list of names, one per line. Names must be in header columns of input files.

--as_only	Optional: Will output the PSI table just to get a sense of alternative splicing. It will use the thresholds for minimum abundance and delta_thresh. It will not perform any statistical analyses.
--html_dir <html_dir>	Optional: location to put html output table and associated images. The images will plot PSI values of sample_set1 against sample_set2. To access the table, open the file <html_dir>/index.html.
--html_out_sign_thresh <float>	Significance threshold of q-value for printed out html_table. DEF=0.05
--pdf	Optional: Will create box images as pdf instead of png as the default.

Additional columns in output

Output follows the PSI table format, but with these additional columns.

Column #	Header	Description
-5	set1_med	Median value in sample set 1
-4	set2_med	Median value in sample set 2
-3	delta_val	set2_med - set1_med
-2	raw_pval	p-value of statistical test
-1	corrected_pval	Corrected p-value of the statistical test

"Percent Spliced In" (PSI) output tables

Most output tables from JuncBASE, regardless of the statistical analysis, gives coordinates for each alternative splicing event and PSI values for every sample examined. Depending on the analysis, there may be additional columns added to the table. The header for the files start with "#".

Column #	Header	Description
1	Contains_Novel_or_only_Known(Annotated)_Junctions	If "N", the event includes a novel splice junction. "K", if all junctions are annotated. The annotation is relative to what was used as the --txt_db3 if (or --txt_db1 by default) in Step 5.
2	as_event_type	Type of alternative splicing event: cassette, alternative_donor (alternative 5' splice site), alternative_acceptor (alternative 3' splice site), mutually_exclusive, coordinate_cassette, alternative_first_exon, alternative_last_exon, jcn_only_AD (alternative 5' splice site, quantified with only junction reads), jcn_only_AA (alternative 3' splice site, quantified with only junction reads)
3	gene_name	Gene(s) associated with the event
4	chr	Chromosome of the event (Note: Fusion genes are not currently supported)
5	strand	"+", "-", or "." if there is ambiguity

6	exclusion_junctions	Coordinates of the first and last position of the intron(s) associated with exclusion isoform(s) of the event. Junctions in distinct isoforms are separated by ";", while grouped junctions from the same isoform are separated by ",". alternative_first_exon-, alternative_last_exon-specific description: Some first/last exons will have the same promoter or transcription termination site, but in addition have alternative donor or acceptor splicing (Further explained in Supplemental Material of the JuncBASE paper). To deconvolve differences between alternative promoter/transcription usage and alternative donor/acceptor usage, I group first(and last) exons that overlap in their genomic coordinates, but differ by an alternative donor(or acceptor) site. The groups are separated by ";" while isoforms within a group are separated by ",". In addition, there are sometimes novel junctions that have no exon model associated with them. The counts for these novel junctions are considered in the exclusion counts. If these novel junctions exist, they will appear first in the ";" separated groups and they will not have corresponding exclusion exons. mutually_exclusive-specific description: Each exclusion exon has a group of two junctions associated with it (the upstream and downstream junction).
7	inclusion_junctions	Coordinates of the first and last position of the intron(s) associated with inclusion isoform of the event. Junctions in distinct isoforms are separated by ";", while grouped junctions from the same isoform are separated by ",". For all alternative splicing event types, only one isoform exists as the inclusion isoform of an event. cassette-specific description:Upstream junctions and downstream junctions are grouped separately. The upstream junctions and downstream junctions are separated by ";", while one or more upstream junction (or downstream junction) is separated by ",". mutually_exclusive-specific description: The upstream and downstream junctions of the inclusion exon are considered one group. The junctions are separated by ",". alternative_first_exon-, alternative_last_exon-specific description: The alternative_first and last exon grouping for the inclusion junctions are explained above for exclusion junctions. For inclusion junctions in alternative first and last exons, they should have a corresponding inclusion exon.
8	exclusion_exons	Coordinates of the first and last position of the exon(s) associated with the exclusion isoform(s). This field is blank for the following events: "cassette" and "coordinate_cassette". Multiple exons are delimited by ";".

9	inclusion_exons	Coordinates of the first and last position of the exon(s) associated with the inclusion isoform(s). Multiple exons are delimted by ";". alternative_donor-, alternative_acceptor-specific description: The inclusion exons are given as the position between each splice site and the next most distal splice site. (Further explained in Supplemental Material of the paper)
10	intron-exon_junctions	Coordinates of the intron/exon boundary. This field is blank for all events EXCEPT for the following events: "alternative_donor","alternative_acceptor", and "intron_retention".
11	neighboring_constitutive_exons	Coordinates of the flanking constitutive exons involved in the event. Multiple exons are delimited by ";".

Columns 12-n, where n is the total number of columns in the table, will depend on the samples analyzed and the analysis. Sample columns will include the sample name and the value will be the PSI calculated for the event. "NA" values indicate either the total expression of the event was low or, if running pairwise statistical analysis, the event was not differentially spliced in the sample.