SplAdder Documentation

Release 3.0.2

Andre Kahles

Feb 07, 2023

Contents:

1 Installation **3**
- 1.1 Install from PyPi . . . 3
- 1.2 Install from source . . . 3

2 General information **5**
- 2.1 The SplAdder directory setup . . . 5
- 2.2 SplAdder is a heuristic . . . 5
- 2.3 Default parameters . . . 6
- 2.4 Working with large datasets . . . 6

3 Run modes **7**
- 3.1 The `build` mode . . . 7
  - 3.1.1 1 Graph construction . . . 8
  - 3.1.2 2 Graph augmentation . . . 8
  - 3.1.3 3 Graph quantification . . . 11
  - 3.1.4 4 Event detection . . . 11
- 3.2 The `test` mode . . . 13
- 3.3 The `viz` mode . . . 14
  - 3.3.1 General organisation . . . 14
  - 3.3.2 Data tracks . . . 15
  - 3.3.3 Order of multiple tracks . . . 16
  - 3.3.4 Plotting range . . . 16
  - 3.3.5 Output names and formats . . . 17
  - 3.3.6 Plotting test results . . . 17

4 Use on large cohorts **19**
- 4.1 1. Single graphs . . . 19
- 4.2 2. Merged graph . . . 20
- 4.3 3. Quantification . . . 20
- 4.4 4. Event Calling . . . 21
- 4.5 General Notes . . . 21

5 File formats **23**
- 5.1 Input Files – `build` mode . . . 23
  - 5.1.1 Annotation Files . . . 23
  - 5.1.2 Alignment Files . . . 23
- 5.2 Output Files – `build` mode . . . 23

5.2.1 Annotation Files in GFF3 Format 24
5.2.2 Event Files in HDF5 Format 24
5.2.3 Event Files in TXT Format 28
5.2.4 Files in PICKLE Format 28
5.3 Output Files – `test` mode 28
5.3.1 Files in TXT Format 28
5.3.2 Diagnose Plots 29
5.3.3 Files in PICKLE Format 30

6 Indices and tables **31**

Welcome to the documentation of SplAdder. Here you will learn more about the details of the SplAdder pipeline, how to apply it to your data, how ^to choose parameters well suited for the kind of question you would like to answer, or how to use and visualize the output data. SplAdder is continuously improved and currently in alpha state. It has been used and tested in context of various research projects of varying scale, ranging from a few samples of *A. thaliana* to thousands of human samples.

To install the latest release, please use:

```
pip install spladder
```

Further information about requirements can be found in the *Installation notes*.

Before diving deeper into the documentation, we suggest to read through the *General information* first.

CHAPTER 1

Installation

There are several ways to obtain SplAdder. You can directly install it from the Python package index using *pip* or you can clone the git repository and set it up yourself.

1.1 Install from PyPi

The installation from Pypi is very straightforward. You can install the latest version of SplAdder using *pip*:

```
pip install spladder
```

If you would like to get a specific version, you can do:

```
pip install spladder=2.2.0
```

1.2 Install from source

If you would like to get the latest changes available on GitHub, you can clone the repository:

```
git clone https://github.com/ratschlab/spladder.git
```

This will create a directory named *spladder* inside your current directory. You can then install SplAdder locally by first changing into the *spladder* directory and then typing:

```
python setup.py install
```

If you are interested in using a specific branch, e.g., *development*, you would need to change into that branch first, before you install. You can achieve this with:

```
git checkout development
python setup.py install
```

CHAPTER 2

General information

In the following, we provide some general information on how the setup of SplAdder works in principle and mention several useful things to keep in mind when using the software.

2.1 The SplAdder directory setup

The general purpose of SplAdder is to build and quantify augmented splicing graphs from RNA-Sequencing data and to utilize them for the detection for alternative splicing events. To achieve this, SplAdder operates on a single *project* at a time, where a project is characterized by a single shared output (or project) directory. All data of subsequent steps will be written to that directory and a certain substructure will directly be created by SplAdder.

SplAdder has different run modes that reflect the different steps of a typical analysis pipeline:

`build` mode for constructing splicing graphs from RNA-Seq data and extracting alternative events

`test` mode for the differential analysis between samples

`viz` mode for the visualization of splicing graphs and alternative events

All of these modes will operate on the same output directory. Please note, that the `build` mode always has to precede the `testing` and `viz` modes, as this creates the splicing graph structures the latter modes operate on.

Please have a look at the *SplAdder run modes* page for further information.

2.2 SplAdder is a heuristic

We would like to reiterate here that SplAdder is a heuristic approach, that employs a system of empirical filter rules on RNA-Seq data to extend a splicing graph pre-defined by a given annotation. Due to this heuristic nature, there is a list of things one should keep in mind when working with the software:

- Although the algorithm is deterministic, it is sensitive to the order of input data. Especially when you are working with many input samples and integrate their information to generate a single splicing graph, the order of input files might influence the outcome. However, in most cases the generated splicing graphs are robust against changes of the order of the input.

- Depending on the annotation file that you are using, some annotated gene regions might be subject to filtering. SplAdder will automatically ignore all introns in the input data that could be assigned to more than one gene in the annotation. (This decision is strand-specific, that is both genes and the intron need to be on the same strand.)
- As SplAdder will sample all possible events from the graph, there can be a certain redundancy in the events (although all events are unique as a whole). For instance, if there are 3 possible donors paired with an acceptor in a gene on the positive strand, SplAdder will output all three possible pairs of alternative 3 prime splice site events.

2.3 Default parameters

For most of the settings available in SplAdder default parameters are assumed. In a basic call in build-mode (`spladder build`), SplAdder requires at least three parameters: the annotation file (via `-a`), a comma-separated list of alignment files (via `-b`) and an output directory where results files are stored (via `-o`):

```
spladder build -o output_directory -b bam_file -a annotation_file
```

This will run SplAdder in its default configuration, which consists of the following steps:

- transform annotation into splicing graph representation
- generate an augmented splicing graph for each alignment file by inferring and adding the following elements:
  - insert intron retentions
  - insert cassette exons
  - insert new intron edges
- merge the augmented splicing graphs into a common splicing graph
- **extract the following alternative splicing events:**
  - exon skip
  - intron retention
  - alternative 3'/5' splice site
  - multiple exon skip
  - mutually exclusive exons
- quantify all alternative splicing events on each of the provided alignment files

2.4 Working with large datasets

SplAdder can be scaled to larger cohorts and has been run successfully on studies including as many as 10000 samples. Working on such large datasets still needs some consideration and planning and might require to call individual steps differently than how it is done for smaller sample sets.

We provide more detailed information in the section describing how to handle *Large cohorts*.

CHAPTER 3

Run modes

SplAdder has different run modes that reflect the different steps of a typical analysis pipeline:

`build` mode for constructing splicing graphs from RNA-Seq data and extracting alternative events

`test` mode for the differential analysis between samples

`viz` mode for the visualization of splicing graphs and alternative events

In the following, we will give a short overview of the different modes and how to use them. Special use cases, for instance the handling of large sample cohorts, will be discussed as a separate topic.

3.1 The `build` mode

The `build` mode is the basic run mode in SplAdder. It is used to construct splicing graphs and to extract alternative splicing events.

To display all available options for `build`, one can simply type:

```
spladder build --help
```

This first step of any SplAdder pipeline consists of several main phases (some of which can be omitted) :

1 Graph construction This is the very initial phase. It parses the given annotation file and summarizes all transcripts of a gene into a splicing graph. This graph will be the basis for all further steps in the workflow.

2 Graph augmentation Given at least one alignment file, the splicing graph of each gene is augmented with new introns and exon segments that were detected in the alignment file. There are different ways how a more than one input alignment files can be combined into final splicing graphs. At the end of this phase, each gene contains an augmented graph that carries not only annotated splice connections but also any novel connections found in the data. Depending on the chosen confidence level, this graph will have a higher or lower density.

3 Graph quantification Once a graph is constructed, all nodes and edges (exons and introns, respectively) in the graph can be quantified using at least one input alignment file. The quantification values can then be used subsequently to quantify splicing events and to compute percent spliced in (PSI) values.

4 Event detection Based on the splicing graph of each gene, SplAdder can detect different types of alternative splicing events: exon skipping, intron retention, alternative 3' splice sites, alternative 5' splice sites, mutual exclusive exons and multiple (coordinated) exon skips. Each event can be quantified using the graph quantifications from the previous step.

In the following, we will provide more in-depth information for each of the phases and describe how the result can be influenced through the choice of command line parameters.

3.1.1 1 Graph construction

This phase runs implicitly before any other phase. We just describe it here for completeness, but in general there is no reason to run this phase only by itself. What it does in the background, though, is to transform the given annotation file:

```
spladder build .. --annotation annotation.gtf ...
```

into a SplAdder specific format, containing all transcript information and the initial splicing graphs per gene. These information will be stored at the same location as `annotation.gtf` and is identified by the suffix `.pickle`. The resulting file in our example would be named `annotation.gtf.pickle`. Depending on the settings, additional files might be created, for instance to mask out certain regions from the annotation. This step is only performed once per annotation file. The summary files will then be re-used by any subsequent SplAdder run using the same annotation file.

The user can influence how SplAdder uses the annotation information in certain situations of ambiguity. However, none of these options is set by default.

In cases of annotations overlapping on the same strand, one can remove the annotation on three different levels.

If two exons of different genes overlap on the same strand, one can remove them with:

```
spladder build ... --filter-overlap-exons ...
```

If two transcripts of different genes overlap on the same strand, one can remove them with:

```
spladder build ... --filter-overlap-transcripts ...
```

If two genes overlap on the same strand:

```
spladder build ... --filter-overlap-genes ...
```

3.1.2 2 Graph augmentation

The augmentation phase brings together alignment file and splicing graphs. Let's assume that you are given an alignment file `alignment.bam` (which should also have an index `alignment.bam.bai`) and an annotation file in GTF format `annotation.gtf`. You can the simply invoke:

```
spladder build --bams alignment.bam \
               --annotation annotation.gtf \
               --outdir spladder_out
```

All three parameters are mandatory for a SplAdder run in `build` mode. Due to the default values of other parameters, this will carry out a full run of all phases. We will describe in the following, which parameters you can change to either only run this phase or to adapt how the splicing graph will be augmented.

Multiple alignment files can be provided using comma-separated notation:

```
spladder build --bams alignment1.bam,alignment2.bam,...
```

Alternatively, a text file, e.g., `alignment_list.txt`, can be provided. This should contain the absolute path to one alignment file per line. The filename has to end in `.txt`. SplAdder can then be invoked with:

```
spladder build --bams alignment_list.txt
```

In its latest version, SplAdder also supports (on an experimental level) CRAM compressed alignment files as input. If you are using such files, in addition to the input filenames of the alignment files, also the path to the indexed reference sequence used for compression is required:

```
spladder build --bams alignment1.cram,alignment2.cram,... --reference path/to/cram_
↪ref.fa
```

Alignment By default, SplAdder only uses primary alignments (in SAM/BAM the ones not carrying the 256 bit-flag). This can be changed by also allowing for secondary alignments to be used:

```
spladder build ... --no-primary-only ...
```

The quality of an alignment is partially determined by the number of mismatches it carries. The default tag in SAM/BAM for this is the `NM:i:` tag. To let SplAdder use a different tag, such as `Nm:i:`, one can use:

```
spladder build ... --set-mm-tag Nm ...
```

Alternatively, one can also force SplAdder not to use any mismatch information (this is not recommended):

```
spladder build ... --ignore-mismatches ...
```

Augmentation Different types of augmentations are possible. The majority of them is switched on by default. For instance the insertion of new intron retentions is always carried out. To switch this step off, one would add:

```
spladder build ... --no-insert-ir ...
```

Similarly, the addition of novel cassette exons is also on by default. To switch this step off, one would add:

```
spladder build ... --no-insert-es ...
```

Also the addition of novel intron edges is switched on by default. To switch it off, one would add:

```
spladder build ... --no-insert-ni ...
```

On the other hand, additional steps for graph cleaning are not switched on by default. For instance the removal of exons shorter than 9nt from the graph can be add with:

```
spladder build ... --remove-se ...
```

Lastly, as SplAdder is a heuristic framework, the addition of novel nodes and edges to the graph depends on the input order of new introns and on the current state of the graph (that is the nodes and edges already present). To increase sensitivity, the addition of new intron edges is iterated a certain number of times (per default 5 times). One can increase the number if iterations, for instance to 10, by:

```
spladder build ... --iterations 10 ...
```

Confidence The confidence level of a SplAdder run determines how strongly input alignments are filtered before new nodes and edges are added to the splicing graphs. In general, there are four confidence levels, with confidence increasing from 0 to 3. The default level is 3 and applies the highest level of filtering. To adapt this choice, e.g., to confidence level 2, one can use:

```
spladder build ... --confidence 2 ...
```

The read filter criteria are dependent on the read length. Here a short overview of the criteria for each of the levels:

Level	Criteria	Value
3	Maximum number of mismatches	0
3	Minimum number of alignments	2
3	Minimum anchor length	ceil(0.25 * readlength)
3	Maximum intron length	350000
2	Maximum number of mismatches	max(1, floor(0.01 * readlength)
2	Minimum number of alignments	2
2	Minimum anchor length	ceil(0.20 * readlength)
2	Maximum intron length	350000
1	Maximum number of mismatches	max(1, floor(0.02 * readlength)
1	Minimum number of alignments	2
1	Minimum anchor length	ceil(0.15 * readlength)
1	Maximum intron length	350000
0	Maximum number of mismatches	max(2, floor(0.03 * readlength)
0	Minimum number of alignments	1
0	Minimum anchor length	ceil(0.10 * readlength)
0	Maximum intron length	350000

In the above table, the *maximum number of mismatches* is used to remove reads that have low quality alignments, the *minimum number of alignments* is the number of split/spliced alignments necessary to confirm a new intron edge for being taken into the graph, the *minimum achor length* is the shortest overlap to an exon segment that a split/spliced alignment needs to have to be counted towards confirming an intron, and the *maximum intron length* is the upper threshold for new introns to be counted.

Merging As SplAdder can be run with multiple alignment files as input, there are several ways on how these files can be combined into forming augmented splicing graphs. This behavior is controlled with the setting of the *merging strategy* using `--merge-strat`.

The first way of merging is to generate a separate augmented splicing graph per given input alignment file. This strategy is called *single* and can be invoked as follows:

```
spladder build ... --merge-strat single ...
```

The second (and default) way of merging is to create a single splicing graph per input file and then merge all graphs into a joint single graph. (This happens for every gene independently.) This strategy is called *merge graphs* and can be invoked as follows:

```
spladder build ... --merge-strat merge_graphs ...
```

A third way of merging is to treat all input alignment files as technical replicates and directly form a splicing graph using all reads. (This makes a difference especially for the count thresholds.) This strategy is called *merge bams* and can be invoked as follows:

```
spladder build ... --merge-strat merge_bams ...
```

The fourth way of merging is a combination of `merge_bams` and `merge_graphs`. In this setting, both steps are performed and both resulting graphs are integrated into a joint graph. The idea behind this setting is to generate maximum sensitivity. However, the improvement is in general marginal and we would not advise to use this setting in general. If you would like to try it nevertheless, you can do so with:

```
spladder build ... --merge_strat merge_all ...
```

Validation SplAdder has the option to validate edges in the graph. This is relevant when working on larger cohorts of samples. In this filtering step an edge is removed if it is not present in the initial annotation and is supported in less than a certain number of input samples. By default this threshold is 10 or the number of input samples in cases where less than 10 samples are given. The threshold can be adapted using `--validate-sg-count`. If nodes get orphaned through the pruning process, they will be also removed from the graph. Following an example that removes all edges from the graph that are present in less than 5 input samples:

```
spladder build ... --validate-sg --validate-sg-count 5 ...
```

3.1.3 3 Graph quantification

In the step of graph quantification, the augmented graph is evaluated again against all given input alignment files, to determine edge and node weights based on the respective expression. If alternative splicing events are to be extracted (next step), this step is carried out automatically. If the user decided not to extract alternative splicing events (explained in the next section), but the graph should be quantified anyways, this can be achieved with:

```
spladder build ... --quantify-graph ...
```

Especially for larger cohorts, it can be challenging to process through all the alignment files for quantification. (We will provide more detailed explanations for this scenario in *Working with large cohorts*.) Here, we will just mention, that the quantification step can be invoked in different modes, called *qmodes*. Let us assume, that two alignment files were provided to SplAdder, `aligment1.bam` and `alignment2.bam`. Then the default is that all files processed sequentially. This quantification mode is called `all` and (despite being used implicitly per default), can also be explicitly set with:

```
spladder build ... --bams alignment1.bam,alignment2.bam \
                   --qmode all ...
```

As an alternative, one can also provide a single alignment file at a time to SplAdder. This strategy is called `single` and can be used to parallelize SplAdder processes across alignment files. It can be invoked via:

```
spladder build .. --bams alignment1.bam --qmode single ...
spladder build .. --bams alignment2.bam --qmode single ...
```

The `single` command always needs to be accompanied by an additional run of SplAdder, that integrates the quantification files for the single alignment files into a joint data structure. For this, all alignment files are provided as input and the quantification mode `collect` is chosen:

```
spladder build .. --bams alignment1.bam,alignment2.bam \
                  --qmode collect ...
```

3.1.4 4 Event detection

In this last phase of the `build` mode, the graphs are used for the extraction of alternative splicing events. Event extraction is performed per default. The user can choose to omit this step entirely (for instance to carry it out at a later point in time). This is done via:

```
spladder build ... --no-extract-ase ...
```

Event extraction SplAdder can currently extract 6 different types of alternative splicing events:

- exon skips (*exon_skip*)
- intron retentions (*intron_retention*)
- alternative 3' splice sites (*alt_3prime*)
- alternative 5' splice sites (*alt_5prime*)
- mutually exclusive exons (*mutex_exons*)
- multiple (coordinated) exons skips (*mult_exon_skip*)

Per default all events of all types are extracted from the graph. To specify a single type or a subset of types (e.g., exon skips and mutually exclusive exons only), the user can specify the short names of the event types (as shown in parentheses above) as follows:

```
spladder build ... --event-types exon_skip,mutex_exons ...
```

In some cases (for instance when integrating hundreds of alignment samples), the splicing graphs can grow very complex. To limit the running time, an upper bound for the maximum number of edges in the splicing graph of a gene to be used for event extraction is set. This threshold is 500 per default. To adapt this threshold, e.g., to 250, the user can specify:

```
spladder build ... --ase-edge-limit 250 ...
```

Event verification Similar to graph validation, SplAdder also performs a step of splice event verification. Only verified events are reported as confident to the user. There are two possibilities how the validity of a confident event is established.

The classical way for event verification is to use heuristic criteria based on the RNA-Seq evidence provided to SplAdder. Depending on the alternative event type, as different set of criteria is used. The tables below summarize the criteria currently in use for the different event types. The order and numbering of criteria is the same as used in the output files of SplAdder.

Multiple Exon Skip	
0	exon coordinates are valid (>= 0 && start < stop && non-overlapping) & skipped exon coverage >= FACTOR * mean(pre, after)
1	inclusion count first intron >= threshold
2	inclusion count last intron >= threshold
3	avg inclusion count inner exons >= threshold
4	skip count >= threshold

Intron Retention	
0	counts meet criteria for min_retention_cov, min_retention_region and min_retetion_rel_cov
1	min_non_retention_count >= threshold

Exon Skip	
0	coverage of skipped exon is >= than FACTOR * mean(pre, after)
1	inclusion count of first intron >= threshold
2	inclusion count of second intron >= threshold
3	skip count of exon >= threshold

Alternative 3/5 Prime	
0	coverage of diff region is at least FACTOR * coverage constant region
1	both alternative introns are >= threshold

Mutually Exclusive Exons	
0	coverage of first alt exon is >= than FACTOR times average of pre and after
1	coverage of second alt exon is >= than FACTOR times average of pre and after
2	both introns neighboring first alt exon are confirmed >= threshold
3	both introns neighboring second alt exon are confirmed >= threshold

In addition to the classical, RNA-Seq evidence based mode, since version 2.5 it is also allowed to use the provided annotation to verify an existing event. In this mode each one of the criteria listed above is replaced with a lookup in the provided annotation. That is, if an intron is already annotated, it will be used for event verification irrespective of any RNA-Seq expression support. This mode is especially useful for single sample analysis, where a complete isoform switch might have occurred and only the alternative event path is supported by reads but not the annotated one. In this case, the event is still reported. This mode is switched off by default and can be activated via:

```
spladder build ... -use-anno-support ...
```

3.2 The `test` mode

This SplAdder mode is for differentially testing the usage of alternative event between two groups of samples. A prerequisite for this is that all samples that are involved in testing have been subjected to a joint analysis in the `build` mode. However, not the full set of samples collected in the `build` mode has to be subjected to testing, but subsets of samples can be used instead.

It is recommended that for each sample condition to be tested (e.g., wild type and some mutant), the number of available replicates is at least three. Further, the mean-variance relationship for intron counts are estimated on the set of tested events. It the number of events to be tested becomes too small, then this estimate becomes unstable and might result in an error.

For the invocation of the testing mode, three different input parameters are mandatory:

```
spladder test --conditionA aligmmentA1.bam,alignmentA2.bam \
              --conditionB alignmentB1.bam,alignmentB2.bam \
              --outdir spladder_out
```

In detail, these are the two lists of alignment files representing the samples for conditions A and B, respectively, as well as the SplAdder output directory. This is the same output directory, as has been used for the `build` mode. Analog to the way a list of alignments can be provided in `build` mode, also in `test` mode the comma-separated file list can be substituted with a file containing the paths to the respective files:

```
spladder test --conditionA alignmentsA_list.txt \
              --conditionB alignmentsB_list.txt \
              --outdir spladder_out
```

By default all event types will be subjected to testing (if they were extracted from the graph prior to testing). If only a specific event type or subset of types should be tested, e.g., exon skips and mutual exclusive exons, the same syntax as in build mode can be applied:

```
spladder test ... --event-types exon_skip,mutex_exons ...
```

If you have built the SplAdder graphs using non-default setting, for instance an adapted confidence level of 2, these parameters also need to be passed in `test` mode, so the correct input files are chosen from the project directory:

```
spladder test ... --confidence 2 ...
```

By default expression outliers are removed in a preprocessing step. If you would like to keep genes that show outlier expression, this behavior can be disabled with:

```
spladder test ... --no-cap-exp-outliers
```

Similarly, you can also switch on the capping of splice outliers, which is not done by default:

```
spladder test ... --cap-outliers ...
```

Sometimes it is useful to assign labels to the two groups being tested, especially is multiple different groupings are analyzed. Groups A and B can be assigned arbitrary labels, such as *Mutant* and *Wildtype*, using:

```
spladder test ... --labelA Mutant --labelB Wildtype
```

In addition, you can also provide a separate tag that will be appended to the output directory name. This is useful, if several rounds of testing or different parameter choices are explored. To tag the output directory with *Round1* you would use:

```
spladder test ... --out-tag Round1 ...
```

The `test` mode is capable of generating several summary plots for diagnosing issues and getting a better understanding of the data being tested. Per default, the plots are generated in *png* format, but other formats such as *pdf* or *eps* can be chosen as well. Per default, the diagnose plots are switched off. To generate them, for instance in *pdf* format, you would use:

```
spladder test ... --diagnose-plots --plot-format pdf ...
```

If several compute cores are available, the computation of the testing can be accelerated by allowing parallel access. If 4 cores should be used:

```
spladder test ... --parallel 4 ...
```

3.3 The `viz` mode

The purpose of this mode is to generate visual overviews of splicing graphs and events and the associated coverage available in the underlying RNA-Seq samples.

3.3.1 General organisation

In general, the plots are organized as individual tracks, which can be stacked to visualize several sources of information jointly. Thereby, the order, number and repetition of tracks can be defined by the user. This allows for the generation of simple overview plots as well as for more complex multi-track visualizations. If more than one track is present, all tracks share the same joint coordinate system on the x axis.

To determine which genomic range is plotted, all elements provided in `--tracks` are considered and a region including all of them is generated. This logic can be overruled using the `--range` parameter to specify a specific range.

However, there are also data track elements that do not necessarily carry any range information (such as a coverage track). In this case the `--range` argument would be required. In the following, we will first explain the definition of tracks in more detail and will then provide some information on how to define a specific range.

3.3.2 Data tracks

This parameter is concerned with defining which data tracks should be visualized in the plot and in which order. The general syntax for specific a data track is as follows:

```
spladder viz --track TYPE [TYPE_INFO [TYPE_INFO ...] ]
```

Here, `TYPE` describes one of the following possibilities (where `TYPE_INFO` is specifically defined for each type):

- **splicegraph** shows the structure of the splicing graph for each of the given genes. If no `TYPE_INFO` is provided, the gene(s) from the `--range` argument are used. To plot the splicing graph for gene with ID *gene1*, one would use:

  ```
  spladder viz --track splicegraph gene1
  ```

- **transcript** shows the structure of all annotated transcripts for each of the given genes. If not `TYPE_INFO` is provided, the gene(s) from the `--range` argument are used. To plot the splicing graph for gene with ID *gene1*, one would use:

  ```
  spladder viz --track transcript gene1
  ```

- **event** shows the structure of the given events, where each event can be specified by its ID. For instance to show the structure of events *exon_skip_2* and *alt_3prime_5*, one can use:

  ```
  spladder viz --track event exon_skip_2 alt_3prime_5
  ```

  If not a specific event ID is given but only the event type, all events of that type for the genes given in `--range` are shown. So to show all *exon_skip* events of gene *gene1*, the correct call would be:

  ```
  spladder viz --range gene gene1 --track event exon_skip
  ```

  If all events of a given gene should be shown, then one can use the special keyword `any` to achieve this:

  ```
  spladder viz --range gene gene1 --track event any
  ```

- **coverage** shows the coverage information in the given range for all samples provided in `TYPE_INFO`. To show coverage for samples *alignment1.bam* and *alignment2.bam*, one would use:

  ```
  spladder viz --track coverage alignment1.bam alignment2.bam
  ```

  If the coverages of both files should be added up, one can also define them as a group:

  ```
  spladder viz --track coverage alignment1.bam,alignment2.bam
  ```

  Sometimes it is useful to assign descriptive labels to single or multiple samples. Given the samples *alignment1.bam* - *alignment4.bam*, which can be separated into groups *wildtype* and *mutant*, respectively, one can use these labels in the plot as follows:

  ```
  spladder viz --track coverage \
                     wildtype:alignment1.bam,alignment2.bam \
                     mutant:alignment3.bam,alignment4.bam
  ```

- **segments** shows the coverage information in the given range as internally used by SplAdder in the splicing graph, quantifying each exonic segment. The usage is analog to `--coverage`.

3.3.3 Order of multiple tracks

The order of the tracks is determined by the order they are provided in at the command line. This is true for both the order of keywords within a single `--track` parameter, as well as for the order of multiple `--track` parameters.

Let us consider the following example:

```
spladder viz --range gene gene1 \
             --track coverage,segments alignment1.bam,alignment2.bam \
             --track event any \
             --track splicegraph \
             --track event exon_skip
```

This plot will have **five** tracks: coverage, segments, events (any), splicing graph, events (only exon skips). This means, even the same track can be plotted multiple times, if requested.

3.3.4 Plotting range

Using the `--range` parameter, the user determines exactly which genomic range is to be considered for plotting the data tracks. This information can be provided as coordinates or the ID information of one or many elements. The usage of `--range` overrules any range determined based on the elements given via `--tracks`. The syntax thereby is as follows:

```
spladder viz --range TYPE TYPE_INFO [TYPE_INFO ...]
```

Here, `TYPE` describes one of the following possibilities (where `TYPE_INFO` is specifically defined for each type):

- **gene** allows for providing at least one gene ID to be considered. If multiple genes should be used, just list them after the `gene` keyword:

  ```
  spladder viz --range gene geneID1 geneID2
  ```

- **event** allows for providing at least one event ID to be considered. If multiple events should be used, just list them after the `event` keyword:

  ```
  spladder viz --range event eventID1 eventID2
  ```

- **coordinate** allows for specifying a coordinate range to be used. Here, the type info contains the list of coordinates to be used. As all ranges will be combined into a joint range eventually, there is little use in providing several coordinate ranges, as the union would be taken. For specifying the genome range of positions 100000 to 101000 on chr1, one would specify:

  ```
  spladder viz --range coordinate chr1 100000 101000
  ```

Note: The `--range` parameter can be used multiple times to combine several ranges. Please note that all provided ranges will be combined into a joint range including all other ranges before plotting. Also note that plotting ranges on different chromosomes is currently not supported as well as plotting ranges exceeding a total length of 1 000 000 bases.

3.3.5 Output names and formats

The user has to choose an output file name for each plot generated. This is specified as the basename of the output file, not containing the output directory or the file ending (which is chosen based on the format). The relevant parameter for this is `--outbase` (or short `-O`). The default format of the plots is *pdf*, but any format supported by Matplotlib can be used. The following two calls for using the output basename *mytest* and the format *png* are equivalent:

```
spladder viz ... --outbase mytest --format png ...
spladder viz ... -O mytest --format png ...
```

Please note that when using the special plotting mode `--test` and providing a test directory with `--testdir` (see below), the plots are not placed in the SplAdder output directory but in the given test directory.

3.3.6 Plotting test results

For visualizing events based on the outcome of the testing mode, there is a special track mode available, which is called `--test`. In principle it works as the other tracks but follows a specific syntax of its elements. There is a default set of 2 tracks that is generated using this option: an event track showing the event of interest and a segements track showing the quantification of segments used for the test for each of the two groups. The general structure is:

```
--test TESTCASE EVENT_TYPE TOP_K
```

While any of the elements is optional, the order is important and elements can only be omitted at the end but not in the middle.

Depending on how the groups are named in testing mode, the output can be found in different subdirectories. So if you groups were *WT* and *MUT*, your output name used by SplAdder would be *testing_WT_vs_MUT*. However, if you did not use any group names, you can just use *default*. Following are two examples for using the default and the specific group mode, respectively:

```
spladder viz ... --test default ...
spladder viz ... --test testing_WT_vs_MUT ...
```

The `EVENT_TYPE` specifies the test result of which event type should be considered. For each of the *k* top events, a separate plot will be generated. You can comma-separate multiple event types or write *any*, for all event types. Here two examples for plotting exon skips and intron retentions or any event, respectively:

```
spladder viz ... --test default exon_skip,intron_retention ...
spladder viz ... --test default any
```

Lastly, the user can specify the number of top events (following the ranking in the testing result file) that should be plotted. If the value is omitted, the default of 1 is used. To plot for instance the top 5 exon skip events, one would use:

```
spladder viz ... --test default exon_skip 5
```

This will create 5 separate plots. The output name will have descriptive suffixes, to tell them apart.

It can happen that the output for testing with SplAdder was written into a user-defined directory and not into the default SplAdder output directory. In this case, the directory can be specified using `--testdir`. For instance, if the test results can be found in *mytestingdir*, the SplAdder call would need to be adapted as follows:

```
spladder viz ... --test default exon_skip 5 --testdir mytestingdir
```

As already noted earlier, this will also influence where the plots for the test are placed. For the above example, all plots will be written to `mytestingdir/plots/`.

Note: If in addition to `--test` further tracks are also defined with `--track`, then each of the tracks is added to **each** of the plots generated for the test results.

CHAPTER 4

Use on large cohorts

While SplAdder is often run on a smaller set of samples, it can also be applied to larger cohorts containing hundreds or thousands of samples. In this setting, it is often advisable to distribute the computation over a high-performance compute cluster and use some workflow management framework (such as Snakemake, Nextflow, or even bash) to coordinate the individual steps.

In the following, we will provide a short overview on how the computation of splicing graphs and their quantification can be split up into individual parts, so they can be run independently. Please note, that the output directory for the whole computation is project specific and stays the same for all (even parallel) runs. SplAdder will not re-compute existing files and when called on different input files, name the outputs accordingly.

The process can be separated into four subsequent logical steps:

1. **Single graphs**: Creating an individual splicing graph per input sample
2. **Merged graph**: Merging all individual graphs into a joint graph representation
3. **Quantification**: Quantifying edges and nodes in the joint graph on each individual input sample
4. **Event Calling**: Calling events (optionally perform testing) on the joint, quantified graph.

For the description of the following steps, we will create a small hypothetical setup of samples, that will be used throughout all commands. Assume that we have a cohort of 10 samples *S1* to *S10*. The aligned (and indexed) bam files for the samples are available as `S1.bam` ... `S10.bam`. We are operating on a given annotation file `annotation.gtf` and our SplAdder project directory that will hold all results will be `spladder_out`.

We will use simple `bash` commands to emulate the distribution of individual tasks. Please note that the code as stated here, would just sequentially compute all single tasks and not generate any parallelization benefit. For this to materialize, you would need to submit the individual tasks to a compute cluster or similar. (If you have a machine with many cores available, you could also trivially parallelize by running several tasks in the background simultaneously.)

4.1 1. Single graphs

In this first step, we will generate a splicing graph for each input sample.

Note: In general a splicing graph is generated per gene. We will abstract from this here and only describe how the graphs will be treated across samples. This implicitly means that this is done per gene.

For each of the given input samples *Si*, we invoke the splicing graph generation separately:

```
for i in $(seq 1 10)
do
    spladder build -o spladder_out \
                   -a annotation.gtf \
                   -b S${i}.bam \
                   --merge-strat single \
                   --no-extract-ase
done
```

This will create an individual splicing graph in `spladder_out/spladder` for each sample. Please note that we added the `--no-extract-ase` option here. This is to prevent SplAdder from automatically proceeding as is done by default in the smaller cohort analyses. With this option present, we gain a more fine-grained controlled over the graph building process. This option will also be present in the subsequent steps.

4.2 2. Merged graph

As a subsequent step, we now need to integrate the graphs across samples, to form one merged splicing graph (per gene). We can just invoke SplAdder again on the same output directory now using a different merging strategy:

```
spladder build -o spladder_out \
               -a annotation.gtf \
               -b S1.bam,S2.bam,S3.bam,S4.bam,S5.bam,S6.bam,S7.bam,S8.bam,S9.bam,S10.
↪bam \
               --merge-strat merge_graphs \
               --no-extract-ase
```

Please note that now all alignment files of the cohort need to be provided. For larger cohorts it is useful to collect all alignment files in a separate files, e.g. `alignments.txt`. Then the merging step could also be invoked as follows:

```
spladder build -o spladder_out \
               -a annotation.gtf \
               -b alignments.txt \
               --merge-strat merge_graphs \
               --no-extract-ase
```

4.3 3. Quantification

Having the merged graph at hand, we can now proceed to quantifying nodes and edges of the graph based on the alignment data. Each quantification will be done independently:

```
for i in $(seq 1 10)
do
    spladder build -o spladder_out -a annotation.gtf -b S${i}.bam \
                   --merge-strat merge_graphs \
                   --no-extract-ase \
                   --quantify-graph \
```

(continues on next page)

(continued from previous page)

```
                    --qmode single
done
```

Please note that now the merging strategy is still `merge_graphs`, as we are quantifying the merged graph and not the individual sample graphs. Also note that we have added the `--qmode single` option.

As a second step to this phase, we need to collect the individual quantifications and aggregate them in a joint database:

```
spladder build -o spladder_out \
               -a annotation.gtf \
               -b S1.bam,S2.bam,S3.bam,S4.bam,S5.bam,S6.bam,S7.bam,S8.bam,S9.bam,S10.
↪bam \
               --merge-strat merge_graphs \
               --no-extract-ase \
               --quantify-graph \
               --qmode collect
```

4.4 4. Event Calling

Now one can proceed analog to the analysis of smaller cohorts. The joint graph is fully quantified and we can move on to use it for downstream analyses, for instance to extract exon skipping events:

```
spladder build -o spladder_out \
               -a annotation.gtf \
               -b S1.bam,S2.bam,S3.bam,S4.bam,S5.bam,S6.bam,S7.bam,S8.bam,S9.bam,S10.
↪bam
               --event-types exon_skip
```

In the above call we have omitted the `--no-extract-ase` option and SplAdder will automatically proceed to this step. As all the intermediate quantification steps are already done, no step will be carried out twice.

4.5 General Notes

When I/O is an issue, SplAdder has the option to generate a compressed summary for each input alignment file. The information contained in that summary is comparable to a wiggle file but has also information on the introns. Using this format will need some additional disk space, but allows SplAdder to perform quantification and querying of intron coverage much more efficiently. You can switch on the use of alignment summaries by:

```
spladder build ... --sparse-bam ...
```

CHAPTER 5

File formats

5.1 Input Files – `build` mode

5.1.1 Annotation Files

SplAdder accepts two different formats for annotation files: GTF and GFF. It will automatically detect the format from the file name ending, so please make sure that your annotation ends with either `gtf` or `gff`. Most sources for genome annotation provide their files in one of these two formats. If you would like to generate your own annotation files, please follow the respective specifications:

GTF: ensembl

GFF: broad, ucsc

5.1.2 Alignment Files

All alignment files are expected to be in BAM format, following the SAM format specification. We have successfully tested SplAdder with the following aligners: - STAR - PALMapper - TopHat

5.2 Output Files – `build` mode

SplAdder produces a variety of different output files. Here we will mainly discuss files that are aimed at the user and omit intermediate files that mainly necessary for internal processes of SplAdder. Most of the latter will be stored in the `spladder` subdirectory in the output directory.

After completing a SplAdder run, you will find several different output files in the output directory. Following, we will describe each file type.

5.2.1 Annotation Files in GFF3 Format

These files have the general pattern `merge_graphs_<event_type>_C<confidence_level>.confirmed.gff3` and contain the events that have been detected by SplAdder. Each event is shown as a mini gene consisting of two different isoforms. If instance an exon skip would be described as:

```
##gff-version 3
Chr1    exon_skip       gene    7616027 7616726 .       +       .       ID=exon_skip.
↪1;GeneName="AT1G21690"
Chr1    exon_skip       mRNA    7616027 7616726 .       +       .       ID=exon_skip.
↪1_iso1;Parent=exon_skip.1;GeneName="AT1G21690"
Chr1    exon_skip       exon    7616027 7616107 .       +       .       Parent=exon_
↪skip.1_iso1
Chr1    exon_skip       exon    7616603 7616726 .       +       .       Parent=exon_
↪skip.1_iso1
Chr1    exon_skip       mRNA    7616027 7616726 .       +       .       ID=exon_skip.
↪1_iso2;Parent=exon_skip.1;GeneName="AT1G21690"
Chr1    exon_skip       exon    7616027 7616107 .       +       .       Parent=exon_
↪skip.1_iso2
Chr1    exon_skip       exon    7616266 7616332 .       +       .       Parent=exon_
↪skip.1_iso2
Chr1    exon_skip       exon    7616603 7616726 .       +       .       Parent=exon_
↪skip.1_iso2
```

For a definition of the different columns, please refer to one of the available GFF3 specifications at ensembl or ucsc. This file will allow you to display the events in a genome viewer such as UCSC Genome Browser, IGV or GBrowse.

5.2.2 Event Files in HDF5 Format

The event files contain all relevant event information and are stored in the hierarchical data format HDF5, allowing for efficient query, addition of data and interoperability between different platforms and languages. You can easily peek into the content of a hdf5 file:

```
$> h5ls -r merge_graphs_exon_skip_C3.counts.hdf5

/                        Group
/conf_idx                Dataset {1}
/confirmed               Dataset {2}
/event_counts            Dataset {4, 7, 2}
/event_features          Dataset {7}
/event_pos               Dataset {2, 6}
/gene_chr                Dataset {1}
/gene_idx                Dataset {2}
/gene_names              Dataset {1}
/gene_pos                Dataset {1, 2}
/gene_strand             Dataset {1}
/num_verified            Dataset {4, 2}
/iso1                    Dataset {4, 2}
/iso2                    Dataset {4, 2}
/psi                     Dataset {4, 2}
/samples                 Dataset {4}
/strains                 Soft Link {/samples}
/verified                Dataset {4, 4, 2}
```

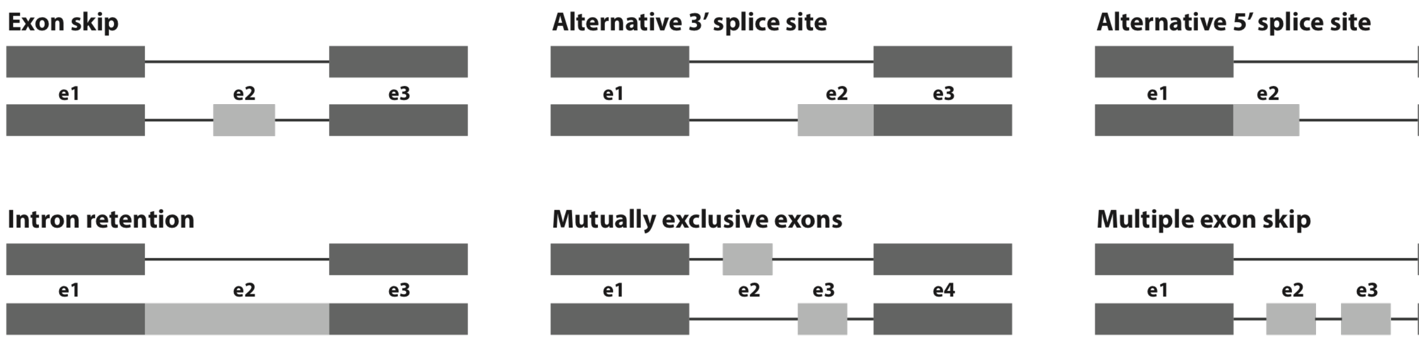
This example lists the contents of a hypothetical exon_skip event hdf5 file, containing the information over 2 exon skips found in a cohort of 4 samples. The tree that is shown looks a little bit like a file system tree and this is also the best analogy to how the file is organized. Directories in the file system would correspond to groups in hdf5 and files in file system to datasets in hdf5. Each group can contain more groups or datasets.

The event hdf5 is structured as follows:

- **conf_idx**: 0-based index set, containing the index of the events that are confirmed in the provided samples
- **confirmed**: binary array indicating for each event whether it is confirmed or not
- **event_counts**: 3-dimensional matrix (S x F x E) containing counts for each of the E events, F features and S samples
- **event_features**: list containing the description of the counted features per event type
- **event_pos**: position of all event exons encoded as start,stop pairs for each event (events are rows, coordinates are columns)
- **gene_chr**: chromosome for each gene in the gene list
- **gene_idx**: index that maps each event to a gene in the gene list (0-based)
- **gene_names**: gene name for each gene in the gene list
- **gene_pos**: position of each gene in the gene list encoded as start,stop pair
- **gene_strand**: strand for each gene in the gene list
- **num_verified**: 2-dimensional count matrix (V x E) containing the number of samples where validation criterion V was met for event E
- **iso1**: 2-dimensional matrix (S x E) containing the number of spliced reads in sample S supporting isoform 1 in event E
- **iso2**: 2-dimensional matrix (S x E) containing the number of spliced reads in sample S supporting isoform 2 in event E:
- **psi**: 2-dimensional matrix (S x E) containing the percent spliced in (PSI) value for event E in sample S. PSI is computed as iso1 / (iso1 + iso2)
- **samples**: names of the samples counted
- **strains**: names of the samples counted (kept for legacy)
- **verified**: 3-dimensional bool matrix (S x V x E) indicating whether validation criterion V for event E was met in sample S

Naming of Features

The naming of features follows a simple logic utilizing the numbering of exon segments as shown in the below image. The numbering follows genomic coordinates. That is the below image shows the positive strand. For the negative strand the numbering would need to be reversed. For instance to count the number of spliced alignments that confirm the connection of exon segments *e1* and *e3* in an exon skip, the corresponding feature name would be *e1e3_conf*.



The below list details the event features for each of the supported event types:

- **features alt3_prime / alt_5prime:**
  - **valid**: contains a 1 if the event is valid and 0 otherwise
  - **e1_cov**: mean coverage of the first constant exon segment in the event
  - **e2_cov**: mean coverage of the exoni segment between the two alternative splice sites
  - **e3_cov**: mean coverage of the second constant exon segment in the event
  - **e1e3_conf**: number of spliced alignments spanning the longer intron
  - **e2_conf**: number of spliced alignments spanning the shorter intron
- **features exon_skip:**
  - **valid**: contains a 1 if the event is valid and 0 otherwise
  - **e1_cov**: mean coverage of the left flanking exon (in genomic coordinates, ignoring strand)
  - **e2_cov**: mean coverage of the cassette exon
  - **e3_cov**: mean coverage of the right flanking exon (in genomic coordinates, ignoring strand)
  - **e1e2_conf**: number of spliced alignments spanning from left flanking to cassette exon
  - **e2e3_conf**: number of spliced alignments spanning from cassette to right flanking exon
  - **e1e3_conf**: number of spliced alignments spanning from left flanking to right flanking exon
- **features intron_retention:**
  - **valid**: contains a 1 if the event is valid and 0 otherwise
  - **e1_cov**: mean coverage of the left flanking exon (in genomic coordinates, ignoring strand)
  - **e2_cov**: mean coverage of the retained intron
  - **e3_cov**: mean coverage of the right flanking exon (in genomic coordinates, ignoring strand)
  - **e1e3_conf**: number of spliced alignments spanning the intron
  - **e2_cov_region**: fraction of positions in the intron that have a coverage > 0
- **features mult_exon_skip:**
  - **valid**: contains a 1 if the event is valid and 0 otherwise
  - **e1_cov**: mean coverage of the left flanking exon (in genomic coordinates, ignoring strand)
  - **e2_cov**: mean coverage over all skipped exons
  - **e3_cov**: mean coverage of the right flanking exon (in genomic coordinates, ignoring strand)
  - **e1e2_conf**: number of spliced alignments spanning from left flanking to cassette exon
  - **e2e3_conf**: number of spliced alignments spanning from cassette to right flanking exon
  - **e1e3_conf**: number of spliced alignments spanning from left flanking to right flanking exon
  - **sum_e2_conf**: number of spliced alignments spanning any of the introns between neighboring skipped exons
  - **num_e2**: number of skipped exons
  - **len_e2**: cumulative length of skipped exons
- **features mutex_exons:**

  - **valid**: contains a 1 if the event is valid and 0 otherwise
  - **e1_cov**: mean coverage of the left flanking exon (in genomic coordinates, ignoring strand)
  - **e2_cov**: mean coverage of the first skipped exon (first defined by genomic coordinates)
  - **e3_cov**: mean coverage of the second skipped exon (second defined by genomic coordinates)
  - **e4_cov**: mean coverage of the right flanking exon (in genomic coordinates, ignoring strand)
  - **e1e2_conf**: number of spliced alignments spanning from left flanking to first exon
  - **e2e4_conf**: number of spliced alignments spanning from left flanking to second exon
  - **e1e3_conf**: number of spliced alignments spanning from first to right flanking exon
  - **e3e4_conf**: number of spliced alignments spanning from second to right flanking exon

Validation Criteria

For each event type, SplAdder uses different empirical validation criteria to determine, whether the called event is valid in a given sample. The overview of that information is provided in the **verified** and **num_verified** fields in the HDF5 count file for each event type. This data also forms the basis for deciding on the list of **confirmed** events. An event is kept as confirmed, if each of the validation criteria is fulfilled in at least one sample. (This does not necessarily mean it is the same sample for different criteria, as the aggregated counts from **num_verified** are used for this decision.) The indices of all confirmed events are indicated in the **conf_idx** array of the HDF5 count file for each event type.

Following, we provide a list of the validation criteria per event type:

- **Multiple Exon Skip**
    1. exon coordinates are valid (>= 0 && start < stop && non-overlapping) & skipped exon coverage >= FACTOR * mean(pre, after)
    2. inclusion count first intron >= threshold
    3. inclusion count last intron >= threshold
    4. avg inclusion count inner exons >= threshold
    5. skip count >= threshold
- **Intron Retention**
    1. counts meet criteria for min_retention_cov, min_retention_region and min_retetion_rel_cov
    2. min_non_retention_count >= threshold
- **Exon Skip**
    1. coverage of skipped exon is >= than FACTOR * mean(pre, after)
    2. inclusion count of first intron >= threshold
    3. inclusion count of second intron >= threshold
    4. skip count of exon >= threshold
- **Alt 3/5 Prime**
    1. coverage of diff region is at least FACTOR * coverage constant region
    2. both alternative introns are >= threshold
- **Mutex Exons**
    1. coverage of first alt exon is >= than FACTOR times average of pre and after
    2. coverage of second alt exon is >= than FACTOR times average of pre and after

3. both introns neighboring first alt exon are confirmed >= threshold
4. both introns neighboring second alt exon are confirmed >= threshold

5.2.3 Event Files in TXT Format

Event files in txt format contain essentially the same information as the HDF5 files in a tab delimited column format with one line per event and the following entries per line:

```
1: chromosome of the event
2: strand of the event
3: unique event_id
4: name of gene the event is located in
5-5+n: start and stop coordinates of the event exons
5+n and following: count values for each of the samples with the following layout␣
↪(features are event type specific as defined above for HDF5 files:
    <sample1>:<feature1>
    <sample1>:<feature2>
    <sample1>:<feature3>
    ...
    <sample2>:<feature1>
    ...
```

The features defined per sample are the same as in the HDF5 files defined above. The number of features thereby depends on the event type.

5.2.4 Files in PICKLE Format

These files are for internal usage only and can be ignored.

5.3 Output Files – `test` mode

In the testing mode, SplAdder generates both tabulated output as well as some images for diagnosing properties of the data. The latter is still in beta mode. Please report an issue on the tracker in case you should encounter any problems.

5.3.1 Files in TXT Format

The results of the `test` mode can be generally found in the `testing` subdirectory of the SplAdder output folder. For each event type {ET} and confidence level {C}, several different output files in text format are generated:

- `test_results_C{C}_{ET}.tsv`
- `test_results_C{C}_{ET}.gene_unique.tsv`
- `test_results_extended_C{C}_{ET}.tsv`

In the following, we will provide more description for each of the files.

Basic test output per event

The basic outputs of testing are stored in the file `test_results_C{C}_{ET}.tsv`. In addition to the header, the file contains one line per tested event. It contains 15 columns carrying the following information:

1. *event_id* – ID of the event

2. *chrm* – event location: chromosome/contig
3. *exon_pos* – event location: exon position (start-stop:start-stop:. . .)
4. *alt_usage* – list of binary values, indicating alternative usage of each exon (same order as in exon_pos)
5. *gene_id* – ID of gene
6. *gene_name* – Name of gene
7. *p_val* – raw p-value from differential test
8. *p_val_adj* – adjusted p-value from differential test
9. *dPSI* – delta PSI (absolute difference between mean-PSI of group A and mean-PSI of group B)
10. *mean_event_count_A* – mean support for tested splice path in group A
11. *mean_event_count_B* – mean support for tested splice path in group B
12. *log2FC_event_count* – log2 fold-change of mean support group A vs group B
13. *mean_gene_exp_A* – mean gene expression of gene in group A
14. *mean_gene_exp_B* – mean gene expression of gene in group B
15. *log2FC_gene_exp* – log2 fold-change of gene expression group A vs group B

Basic test output per gene The file `test_results_C{C}_{ET}.gene_unique.tsv` contains essentially the same information as the basic test output per event, just made unique per gene. That is, if a gene contains multiple events of the same type, here only the most significant one is reported. The columns are the same.

Extended test output per event The file `test_results_extended_C{C}_{ET}.tsv` contains additional output for each tested event and can be used for debugging purposes. The number of columns is variable and depends on the size of the input groups used for testing. For the following explanation, we assume that input group A has size 2 and input group B has size 3. The first 15 columns are identical to the basic event output file. The additional columns are as follows:

16. *event_count:group_A_sample1* – support for tested splice path in group A sample 1
17. *event_count:group_A_sample2* – support for tested splice path in group A sample 2
18. *event_count:group_B_sample1* – support for tested splice path in group B sample 1
19. *event_count:group_B_sample2* – support for tested splice path in group B sample 2
20. *event_count:group_B_sample3* – support for tested splice path in group B sample 3
21. *disp_raw* – raw dispersion estimate for the tested event
22. *disp_adj* – corrected dispersion estimate for the tested event

5.3.2 Diagnose Plots

The testing mode can generate some diagnose plots (via `--diagnose-plots`) that can help you assess the data you are looking at. These plots are still in beta mode and might change in future versions of SplAdder.

The plots reside in the SplAdder output directory in the folder `testing/plots`. Currently, the following plots are available:

Count distribution A plot showing the distribution of supporting counts and the gene expression over events per tested group / condition. The plot is available over raw counts and over log10 transformed counts.

MA plot A plot showing the log2 fold-change of each event over the mean normalized counts.

Dispersion Three plots showing the raw dispersion estimate, the dispersion fit and the adjusted dispersion..

QQ plot Quantile-quantile plots showing the distribution of p-values after testing over a uniform distribution to check for over-inflation. Available for raw and adjusted p-values.

5.3.3 Files in PICKLE Format

Similar to `build` mode, these files are for internal usage only and can be ignored.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search