splicekit

splicekit: a toolkit for splicing analysis from short-read RNA-seq

An integrative platform for splicing analysis of RNA-seq short-read sequencing data. **splicekit** input are read alignments in BAM format (look at datasets for details on how to run examples).

splicekit is a modular platform for splicing analysis of RNA-seq datasets. The platform also integrates an JBrowse2 instance together with pybio for genomic operations and scanRBP for RNA-protein binding

studies. The whole analysis is self-contained (one single folder) and the platform is written in Python, in a modular way.

What is splicekit?

From an initial config file (`splicekit.config`), sample annotation (`samples.tab`) and aligned reads in BAM format, **splicekit** first defines comparisons (which test samples to compare to which controls). Next, feature count tables are generated (exons, anchors, junctions, genes) based on defined comparisons. Analysis incude edgeR alt-splice (differentially used features), motif analysis with DonJuAn (junction-anchor) and DREME, RNA-protein binding enrichment analysis with scanRBP and clustering analysis on expression of features. To facilitate result and data interpretation, splicekit also provides an instance of JBrowse2.

Installation

The easiest way to install splicekit is to simply run:

```
Unset
pip install splicekit
```

Note that on some systems, **pip** is installing the executable scripts under `~/.local/bin`. However this folder is not in the PATH which will result in command not found if you try to run `splicekit` on the command line. To fix this, please execute `export PATH="$PATH:~/.local/bin"` (and add this to your `~/.profile` to make it run on every login). Another suggestion is to install inside a virtual environment (using virtualenv).

To install splicekit directly from this repository:

```
Unset
pip install git+https://github.com/bedapub/splicekit.git@main
```

Quick start

To start the analysis with **splicekit**, you need the following:

1. install and process the reference genome of choice with pybio
    a. if you installed splicekit with pip, then this will also install the pybio dependency
    b. you can download and process your reference species genome with
        i. example for *Homo sapiens*: `pybio genome homo_sapiens`
    c. to search for other species, simply run `pybio search species`
2. aligned reads stored in **BAM files** (one file per sample)
    a. you can simply align your FASTQ sequence files to the reference genome using the provided example script for alignment; the scripts downloads the reference genome of choice with pybio and aligns the reads with STAR

3. `samples.tab` file: containing one line per sample; this file connects the BAM files to the sample_id and also defines treatment; for an easier start, look at one of the provided samples.tab in the datasets;
4. `splicekit.config` file: here you define basic splicekit parameters, like the reference genome, the folder where BAM files are stored etc.; again you can start with one of the provided splicekit.config files

There are four example folders in the dataset collection. Each of these folders contains a full set of scripts to download and process publicly available RNA-seq datasets from scratch.

splicekit.config

Here we describe the parameters in the splicekit.config configuration file. We subset the parameters into several tables, depending on their scope.

Core parameters

`study_type`, `default = "my_study"`
Descriptive short study name, arbitrary string describing the study.

`library_type`, `default = "paired-end"`
Possible values:

- "SECOND_READ_TRANSCRIPTION_STRAND"
- "FIRST_READ_TRANSCRIPTION_STRAND"
- "SINGLE_STRAND"
- "SINGLE_REVERSE"
- "NONE"

For unstranded data, we enter "NONE". For paired-end stranded data, the most common is "SECOND_READ_TRANSCRIPTION_STRAND", which means the second read of the pair maps in the transcript direction, and the first read maps in the reverse direction. For stranded single-end sequencing, "SINGLE_STRAND" means the reads map in the transcript direction, and "SINGLE_REVERSE" means the reads map on the opposite strand of the transcripts. Also for single-end unstranded we specify "NONE".

Sample annotation parameters

`sample_column`, `default = "sample_id"`
This parameter tells splicekit **which column** to look for in the **samples.tab** file for **sample IDs**. BAM file names are then constructed with `sample_id.bam`

`treatment_column`, `default = "treatment_id"`
Which column from **samples.tab** defines treatment (test and control labels).

`control_name`, `default = "DMSO"`
By which text control samples are identified in the treatment column in samples.tab. Non-control samples are then compared to these marked controls.

`separate_column, default = “”`
Separate comparisons by grouping samples on separate_column? Do not separate by default (empty parameter ""). This is useful if, for example, you have samples from two different tissues (let’s say tissue A and B), and you would only like to do comparisons inside each tissue (tissue A controls vs tissue A treated samples, and another comparison with tissue B controls vs tissue B treated). By introducing a column in samples.tab marking the tissue and providing the column name as the parameter for separate_column, splicekit will only compare samples inside the same tissue in this example.

`group_column, default = “”`
Only include controls in the same “domain” with other samples. If, for example, the samples are coming from sequencing on plate1, plate2 and plate3, then also only control samples from the same plates will be considered for the comparison.

Genome specific parameters

`species, default = None`
Genome species, this is connected with pybio and Ensembl. In general, pybio knows about all the genome species in Ensembl, however if you added your own custom species via pybio and a FASTA+GFT file, this can also be specified here.

`genome_version, default = “homo_sapiens”`
Ensembl genome version or custom genome version, leaving None will take the latest Ensembl version.

`bam_path, default = “bam”`
The folder where BAM files are stored. These files will be used by splicekit and are expected to be named with `sample_id.bam`. The folder location can be absolute or relative to the splicekit folder (the working folder where you have your splicekit.config and samples.tab files).

scanRBP parameters

`scanRBP, default = True`
Should splicekit run the scanRBP part of the analysis? If **True**, it will be run, if **False**, it will be skipped.

`protein, default = “TARDBP.K562.00”`
This is the ID (name) of the protein PWM for scanning sequences identified as regulated by splicekit. To see a list of proteins (and their PWM identifiers), you can run `scanRBP search search_term`. An example for hnRNPA would be:

```
Unset
# search for available protein PWM IDS for hnRNPA
scanRBP search hnRNPA

          scan_id protein tissue                                    description
HNRNPA1.HepG2.00 HNRNPA1  HepG2 heterogeneous nuclear ribonucleoprotein A1
HNRNPA1.HepG2.01 HNRNPA1  HepG2 heterogeneous nuclear ribonucleoprotein A1
HNRNPA1.HepG2.02 HNRNPA1  HepG2 heterogeneous nuclear ribonucleoprotein A1
```

```
HNRNPA1.HepG2.03 HNRNPA1  HepG2 heterogeneous nuclear ribonucleoprotein A1
HNRNPA1.HepG2.04 HNRNPA1  HepG2 heterogeneous nuclear ribonucleoprotein A1
HNRNPA1.HepG2.05 HNRNPA1  HepG2 heterogeneous nuclear ribonucleoprotein A1
 HNRNPA1.K562.00 HNRNPA1   K562 heterogeneous nuclear ribonucleoprotein A1
 HNRNPA1.K562.01 HNRNPA1   K562 heterogeneous nuclear ribonucleoprotein A1
 HNRNPA1.K562.02 HNRNPA1   K562 heterogeneous nuclear ribonucleoprotein A1
 HNRNPA1.K562.03 HNRNPA1   K562 heterogeneous nuclear ribonucleoprotein A1
 HNRNPA1.K562.04 HNRNPA1   K562 heterogeneous nuclear ribonucleoprotein A1
```

`protein_label`, `default = "TDP43"`
This is just a short descriptive label for the protein used. This label will be used in graphs, reports etc.

Processing parameters

`platform`, `default = "desktop"`
This parameter tells splicekit if to run jobs on a single computer (`"desktop"`) or to submit jobs to a LSF cluster with bsub (`"cluster"`).

`container`, `default = ""`
Possible values:

- "" empty string, all software dependencies are expected to be installed on the machine / cluster
- `"singularity run docker://ghcr.io/bedapub/splicekit:main"`

If left empty, **splicekit** will assume you have all software dependencies installed on your machine or cluster. An alternative is for you to have **singularity** available and you would then set the container parameter to `"singularity run docker://ghcr.io/bedapub/splicekit:main"`. This will then automatically download and import a Docker image that we provide and run all software (except pybio and scanRBP, which are already installed as dependencies via pip) from this imported container.

Visualization and labeling parameters

`short_names`, `default = "[]"`
By default, no names are shortened or replaced in the results and processing steps of **splicekit**. However, sometimes it is convenient to replace long strings with shorter identifiers. This is possible by providing a list of triples which represent rules for replacing / shortening names.
Example: `[("cell_line_A", "A", "complete")]`, this would only replace `cell_line_A` with `A` if `cell_line_A` is the whole string, if you specify `"partial"`, also strings like `...cell_line_A...` would be replaced with `...A...`

```
Unset
# example splicekit.config short_names parameter
# replace cell_line_A with A
```

```
# replace cell_line_B with B
short_names = '[("cell_line_A", "A", "complete"), ("cell_line_B", "B",
"complete")'
```

After setting the basic parameters in your splicekit.config file, simply run splicekit process inside the folder with your config file. This will run all available analysis on your dataset.

Running splicekit

For example files (`splicekit.config` and `samples.tab`) please look in the datasets folder.

If you have a folder with splicekit.config and samples.tab, a comprehensive run of all analysis is incorporated in a single splicekit command: `splicekit process`. Each analysis step can be run separately by a single splicekit command.

Unset

```
splicekit process      # runs all analysis
splicekit setup        # initializes folder structure
splicekit annotation   # loads samples.tab annotation and creates comparisons
splicekit features     # prepares count tables in data/* folders
splicekit edgeR        # runs edgeR analysis on the cluster
splicekit motifs       # runs motif analysis together with scanRBP
splicekit promisc      # runs promisc analysis from edgeR results
splicekit judge        # generates juDGE plots
splicekit clusterlogfc # generates cluster of pair-wise logFC comparisons
splicekit jbrowse2     # starts JBrowse2 visualization with local web server
splicekit report       # creates html report in folder report
splicekit version      # prints out current version
```

Annotation and comparisons: splicekit annotation (samples.tab)

This first step of the analysis (`splicekit annotation`) loads samples from the file `samples.tab`. It also considers the `treatment_column`, `control_name`, `group_column` and `separate_column` to create comparisons. Each treatment (can have several replicates / samples / readouts) is compared to the controls. The comparisons are stored in the file `annotation/comparisons.tab`.

Note: the file samples.tab is TAB delimited.

sample_id	treatment_id
sample1	control
sample2	control
sample3	test1
sample4	test1

```
sample5     test2
sample6     test2
```

splicekit would then expect BAM files with names `sample_id.bam` to be present (sample1.bam, sample2.bam, etc.) in the folder `bam_path` parameter specified in the `splicekit.config` file.

The created comparisons will by default compare treated samples to control samples. The comparisons are also stored in a tab delimited file (`annotation/comparisons.tab`), for the above example:

comparison	compound_samples	DMSO_samples
test1_control	sample3_test1,sample4_test1	sample1_control,sample2_control
test2_control	sample5_test2,sample6_test2	sample1_control,sample2_control

In addition, this step will also create processing shell scripts and cluster job files (jobs/*). An example cluster job file:

Unset

```
#!/bin/bash
#BSUB -J edgeR_junctions_sample1                  # job name
#BSUB -n 4                                        # number of tasks
#BSUB -R "span[hosts=1]"                          # 1 host
#BSUB -q short                                    # select queue
#BSUB -o logs_edgeR_junctions/sample1_control.out # output file
#BSUB -e logs_edgeR_junctions/sample1_control.err # error file

ml R
R --no-save --args splicekit comparison_junctions_data junctions control test
... < comps_edgeR.R
```

Features count tables: splicekit features

Running splicekit features will create count tables for junctions, anchors, exons and genes.

What are features?

splicekit operates with 4 types of features: **junctions, anchors, exons and genes**. All feature IDs have the same format: `chrstrand_start_stop`. An example would be `chrX-_154371360_154374505`. Please check genomic coordinates for an explanation of how we report genomic coordinates across splicekit.

Feature data files

In the folders `data/sample_{feature_type}_data`, each individual file (table) contains the list of all features and the count for the individual sample.

Example file would be `data/sample_exons_data/sample_99.tab`:

GeneID	Start	End	Length	Symbol	1_test	2_test	3_control	4_control
1	58347029	58347353	325	A1BG	42	31	109	75
1	58347640	58350370	2731	A1BG	0	0	3	1
1	58350651	58351391	741	A1BG	0	0	10	1

Running edgeR analysis on features: splicekit edgeR

Running edgeR analysis on features (junctions, anchors, exons, genes) consist of simply running `splicekit edgeR`.

Results are stored in the files `results/results_edgeR_{feature_type}.tab` where feature_type is one of ["genes", "exons", "junctions", "donor_anochors", "acceptor_anchors"]. Only results with `FDR<splicekit.config.edgeR_FDR_thr` are reported (sorted by FDR), linked to JBrowse2 with URL links.

To explore all results (no FDR filters), look at files: `results/results_edgeR_{feature_type}_all.tab`.

Motif analysis: splicekit motifs

Motif analysis on donor site patterns (9nt sequences) is performed on the top 100 hits for each comparison. The analysis is run by `splicekit motifs`. Html reports are generated at results/motifs. In addition to computing motif logos, we run **DREME** on regulated sequences vs. controls.

scanRBP

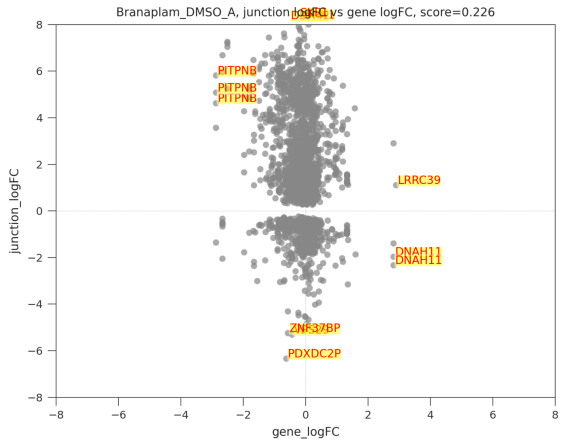
The RNA-protein binding analysis is run as part of the motif analysis above: to identify potential enrichment of RNA-protein binding at regulated sites (donor sites, acceptor sites, other areas), we developed and integrated scanRBP with splicekit.

Once we identify sets of control and regulated sequences, we compute log-odds of the binding signal for a specific protein of interest from its PWMs. Performing bootstraps on the sequence labels, we can estimate the probability the binding at regulated sequences is different from the control binding (log-fc of the binding signal).

scanRPB is integrated with splicekit and can also be used separately as an individual package (`pip install scanRBP`).

juDGE plots

To determine the effect of a treatment (more involved in gene expression changes in general or more involved in splicing changes) we can generate juDGE plots (junction logFC vs. gene logFC). These plots are stored in `results/judge/*` and contain PNG images and also html interactive plotly reports (mouse over shows data and gene name).



In the plot above, the left panel represents a comparison where the compound causes mostly changes in junctions of several genes (y axis) and junctions are much more perturbed compared to gene expression in general (activity on the x axis). This compound would be characterized as a "**splicing modifier**".

In contrast, the comparison on the right panel shows more activity on changing gene expression in general (x axis). The compound involved would be labeled as an "**expression modifier**".

JBrowse2 visualizations

To graphically explore results, splicekit provides an integrated JBrowse2 wrapper. There are two steps in providing JBrowse2 visualizations:

```
Unset
# process JBrowse2 files (genome, BAM, etc.)
splicekit jbrowse2 process

# start local web server with created JBrowse2 config file
splicekit jbrowse2 start

# note: running "splicekit jbrowse2" will run both steps above
# note: running "splicekit process" also runs jbrowse2 steps
```

splicekit jbrowse2.

Documentation and specifications

Genomic coordinates

All genomic coordinates splicekit operates with are 0-based left+right inclusive. E.g. the range 100-103 would include coordinates 100, 101, 102 and 103. The first coordinate is 0. More specific details:

- **feature specific**: all feature coordinates (junctions, anchors, exons) are given in a numeric sort order regardless of strand (feature_start is always < feature_stop). Example chr1+_100_102 would represent a junction spanning coordinates [100,101,102]). Example chr1-_100_102 would represent a junction spanning coordinates [102,101,100])
- **junction specific**: junction coordinates cover/overlap 1-nucleotide of adjoining exons. Example 1: chr1+_100_200 represents a junction on chromosome 1 (+ strand) from [100..200]. 100 is the last nucleotide of the upstream exon and 200 is the first nucleotide of the downstream exon. Example 2: chr1-_100_200 represents a junction on chromosome 1 (- strand) from [100..200]. 200 is the last nucleotide of the upstream exon and 100 is the first nucleotide of the downstream exon.

Important

Refseq and Ensembl GTF files are 1-indexed. When splicekit reads files from refseq/ensembl, it performs `coordinate -= 1` on all coordinates to keep this in line with other internal coordinate structures (which are all 0-indexed).

File reference/junctions.tab

This file contains all the junctions detected from all the samples in the project. Only junctions that could be annotated to genes are reported. However, "novel" junctions (that do not touch refseq/ensembl annotated exons) are also reported, as long as the start and stop of the junction falls inside an annotated gene (see annotated column).

`junction_id`, `example = "chr1+_17741_17839"`
Unique ID of the junction in format: chrstrand_start_stop.

`donor_anchor_id`, `example = "chr1+_17725_17740"`
Matching donor_anchor_id, by default 15nt region upstream of junction start.

`acceptor_anchor_id`, `example = "chr1+_17840_17855"`
Matching acceptor_anchor_id, by default 15nt region downstream of junction stop.

`gene_id`, `example = "ENSG00000120948"`
Ensembl or Refseq gene_id; note that a junction can be non-annotated (column annotated!="AA") but still assigned to a gene, meaning its start and stop are inside the gene.

`gene_name`, `example = "TARDBP"`
Corresponding to gene_id.

`chr`, `example = "1"`
Chromosome of the junction and anchors.

`strand`, `example = "+"`
Strand of the junction and anchors. Either + (plus) or - (minus).

`annotated`, `example = "AA"`

Two letter code: AA / AN / NA / NN; first letter related to donor site (5' of junction), second letter related to acceptor site (3' of junction); A = annotated (touches annotated exon), N = not annotated

`count`, `example = "553"`
Integer raw count of all reads across all samples in the project that support the reported junction.

Files reference/donor_anchors.gtf and reference/acceptor_anchors.gtf

GTF files generated from all the donor/acceptor anchors in the `reference/junctions.tab` file. These GTF files are then used by **featureCounts** to create tables of counts for anchors across project samples.

Files results/results_edgeR_{feature_type}.tab

The general structure of the edgeR results files for all features is the same, with specific columns added for specific features (see below the table for further details). Results are filtered by `splicekit.config.edgeR_FDR_thr`.

General columns

`result_id`, `example = "r1"`
r_int identifier of result, starts with 1

`comparison`, `example = "test_control"`
Name of the comparison, taken from `annotation/comparisons.tab`.

`compound`, `example = "treatment1"`
Name of the treatment / compound tested.

`feature_id`, `example = "chr1+_17741_17839"`
The ID of the reported feature. Could be: gene_id / exon_id / junction_id / [donor, acceptor]_anchor_id

`chr`, `example = "1"`
Chromosome of the feature.

`strand`, `example = "+"`
Strand of the feature. Either + (plus) or - (minus).

`feature_start`, `example = "17741"`
Start of feature (numerically, start always < stop), also see genomic coordinates.

`feature_stop`, `example = "17839"`
Stop of feature (numerically, stop always > start), also see genomic coordinates.

`feature_length`, `example = "250"`
Length of feature (feature_stop - feature_start + 1).

`gene_id`, `example = "ENSG00000120948"`
Ensembl or Refseq gene_id.

`gene_name, example = "TARDBP"`
Corresponding to gene_id.

`sum_feature_test, example = "1000"`
Sum of counts for this feature across all test samples.

`sum_feature_control, example = "1000"`
Sum of counts for this feature across all control samples.

`jbrowse_loc, example = "3:342321..351243"`
Genomic region that will be displayed in JBrowse.

`jbrowse_url, example = ""`
Link to JBrowse view.

`logFC, example = ""`
log fold change, reported from edgeR.

`exon.F, example = ""`
exon.F, reported from edgeR.

`p_value, example = ""`
p_value, reported from edgeR.

`fdr, example = ""`
false discovery rate, reported from edgeR.

Adding to the above column table, there are additional columns present, depending on the feature.

Junction specific (additional) columns

`annotated, example = "AA"`
Two letter code: AA / AN / NA / NN; first letter related to donor site (5' of junction), second letter related to acceptor site (3' of junction); A = annotated (touches annotated exon), N = not annotated.

`donor_anchor_id, example = ""`
ID of the donor_anchor, associated / linked to this junction.

`acceptor_anchor_id, example = ""`
ID of the acceptor_anchor, associated / linked to this junction.

`UTR, example = ""`
Contains text first_exon_{start_pos}, if the junction touches any first exon of any transcript of the gene.

Exon specific (additional) columns

`delta_PSI, example = ""`

Value for test_PSI-control_PSI, percentage spliced-in.

Issues and Suggestions

Use the GitHub repository issues page to report issues and leave suggestions.