KisSplice
De-novo calling alternative splicing events from RNA-seq data

User's guide, version 2.6.7

2026-09-02

Contents

1 KisSplice, at a glance **2**

2 KisSplice software package **2**
- 2.1 CeCILL licence . . . 2
- 2.2 Reference . . . 2
- 2.3 Online ressources . . . 2
- 2.4 Installation . . . 2
  - 2.4.1 Debian/Ubuntu package . . . 2
  - 2.4.2 Conda . . . 3
  - 2.4.3 Docker . . . 3
  - 2.4.4 Binary release for Debian/Ubuntu . . . 3
  - 2.4.5 Compile from source . . . 3

3 Usage **4**
- 3.1 Testing KisSplice in a couple of minutes . . . 4
- 3.2 Common TroubleShooting . . . 5
- 3.3 Options . . . 6
- 3.4 Counts option . . . 6
- 3.5 Paired-end reads . . . 7

4 I/O **7**
- 4.1 Input . . . 7
- 4.2 Output . . . 7
- 4.3 Uncoherent Bubbles . . . 9

5 Performances **9**
- 5.1 Pre-treatment . . . 9
- 5.2 Job calibration . . . 9
- 5.3 Relative coverage . . . 10
- 5.4 Big datasets . . . 10

6 Further questions **10**

1 KisSplice, at a glance

KisSplice is dedicated to *de-novo* calling of alternative splicing events from one or several RNA-seq datasets. In addition to splicing events, KisSplice detects indels and SNPs. Data from different conditions can be co-assembled with KisSplice.

2 KisSplice software package

2.1 CeCILL licence

This software is governed by the CeCILL licence. Details are mentioned in the COPYING file.

2.2 Reference

If you use KisSplice in a published work, please cite the following reference:
Sacomoto et al. KISSPLICE: de-novo calling alternative splicing events from RNA-seq data. BMC Bioinformatics 13, S5 (2012). https://doi.org/10.1186/1471-2105-13-S6-S5

If you use KisSplice, kissplice2reftranscriptome and kissDE, please cite:
Lopez-Maestre et al., SNP calling from RNA-seq data without a reference genome: identification, quantification, differential analysis and impact on the protein sequence, Nucleic Acids Research, Volume 44, Issue 19, 2 November 2016, Page e148

If you use KisSplice, kissplice2refgenome and kissDE, please cite:
Benoit-Pilven et al. Complementarity of assembly-first and mapping-first approaches for alternative splicing annotation and differential analysis from RNAseq data. Sci Rep 8, 4307 (2018). https://doi.org/10.1038/s41598-018-21770-7

2.3 Online ressources

The main repository is at https://gitlab.inria.fr/erable/kissplice. It contains releases, an issue-tracker for questions or bug reports, and allows people to contribute.

Biostars forum: www.biostars.org/t/kissplice/ (seems unused).

2.4 Installation

KisSplice is written in C/C++/Python and is running on Mac OS X and Linux 64 bits platforms. It can be installed using the following methods, by decreasing order of convenience.

2.4.1 Debian/Ubuntu package

KisSplice is available as a system package in Debian/Ubuntu.

```
apt install kissplice
kissplice --help
```

2.4.2 Conda

Kissplice is available in conda in the bioconda channel. Bioconda creates packages for Linux (x86_64 and arm64) and MacOS (x86_64 = intel macs before M1).

2.4.3 Docker

You can find the latest version of KisSplice, KisSplice2RefGenome and KissDE on Docker Hub.

We also propose a stand-alone Docker image for the KissDE Shiny App for KisSplice results exploration.

2.4.4 Binary release for Debian/Ubuntu

A precompiled standalone binary package for ubuntu is available in the release page.

```
# Decompress downloaded archive
tar xf kissplice-binary-ubuntu-<version>.tar.gz
# Access the kissplice tool
kissplice-binary-ubuntu-<version>/bin/kissplice --help
```

2.4.5 Compile from source

Required dependencies :

- cmake >= 3.9
- C/C++14 compiler toolchain (binutils, gcc or clang)
- python3 to run kissplice
- git

Download a source code archive from the latest release and uncompress it.

```
# directory where the release tar.gz was uncompressed
cd kissplice/

# Replace install_directory by the path where you want your install to be.
# If you have latex installed you can add -DUSER_GUIDE=ON to generate
# the user guide at install_directory/doc/user_guide.pdf
cmake -S . -B build/ -DCMAKE_BUILD_TYPE=Release -DUSE_BUNDLED_BCALM=ON
    -DCMAKE_INSTALL_PREFIX=install_directory

cmake --build build/ # add -jN to use up to N cpu cores in parallel
cmake --install build/

# Example of use
install_directory/bin/kissplice -r sample_example/mock1.fq -r sample_example/mock2.fq
    -r sample_example/virus1.fq -r sample_example/virus2.fq
```

If you want to build from source and install to a local directory (like `/.local/`):

```
# directory where the release tar.gz was uncompressed
# or where the git was cloned
cd kissplice/
cmake -S . -B build/ -DCMAKE_BUILD_TYPE=Release
    -DCMAKE_INSTALL_PREFIX=<install directory>
cmake --build build/
cmake --install build/
```

3 Usage

3.1 Testing `KisSplice` in a couple of minutes

The example presented here only deals with alternative splicing events, but the format is the same for SNPs and indels.

The `sample_example` directory contains four files `mock1.fq`, `mock2.fq`, `virus1.fq`, `virus2.fq` which correspond to RNAseq reads from the MBTD1 gene in two biological conditions: mock infection and infection by Influenza A virus. There are two biological replicates for each condition. Running `KisSplice` on this dataset should output one splicing event.

```
./bin/kissplice -r sample_example/mock1.fq -r sample_example/mock2.fq
 -r sample_example/virus1.fq -r sample_example/virus2.fq
less results/results_mock1_mock2_virus1_virus2_k41_coherents_type_1.fa
```

For our example, the output files can be found in the results directory: `results_mock1_mock2_virus1_virus2_k41_coherents_type_1.fa` which contains the alternative splicing event (Fig. 1). The other files are empty.

```
>bcc_1|Cycle_0|Type_1|upper_path_length_226|AS1_15|SB1_23|S1_9|ASSB1_0|AS2_36|SB2_2
6|S2_16|ASSB2_0|AS3_8|SB3_6|S3_4|ASSB3_0|AS4_6|SB4_6|S4_1|ASSB4_0|rank_0.55963
TGAGGTAGTCAAACCATTTAAAAGGAAGTTTTGTGTAACCTCTGGGTGGAGTAAGTTCAATCATGTTAATTTCACAGAAACCG
ACAGGGAAAATAGAAGGAGAGGTTGCATGGTAACAGAACCAGTCAGATCCGTCTGCTGCTTCTGAGCCATCGATCCCAATCAT
CAGGAATCCGTCAGCTAGCACCTTTCTAATGGTTGCGACACATATTGTAGAAAGATTTAA
>bcc_1|Cycle_0|Type_1|lower_path_length_78|AB1_2|AB2_2|AB3_14|AB4_15|rank_0.55963
TGAGGTAGTCAAACCATTTAAAAGGAAGTTTTGTGTAACCTTTCTAATGGTTGCGACACATATTGTAGAAAGATTTAA
```

Figure 1: Content of the file `results_mock1_mock2_virus1_virus2_k41_coherents_type_1.fa`, the alternative splicing event found. In red are the flanking k-mers (i.e. extremities of the flanking exons). In white is the variable part (i.e. the skipped exon).

The output is organised in blocks of four lines, each corresponding to an AS event. Here there is just one block. Each block is composed of two sequence variants. The upper path corresponds to the longer variant, and the lower path to the shorter variant. The sequence of each path starts and ends with the k-mer common to both path (highlighted in red). The upper path contains an additional sequence (highlighted in white). In this example, the additional sequence is an alternatively skipped exon. The flanking k-mers are the extremities of the flanking exons.

The fasta header of each path additionnally contains read counts. When using the –counts 0 option (Figure 2), the counts are easier to understand because there is just one count for each input file.

```
>bcc_1|Cycle_0|Type_1|upper_path_length_226|C1_47|C2_78|C3_18|C4_13|rank_0.55963
TGAGGTAGTCAAACCATTTAAAAGGAAGTTTTGTGTAACCTCTGGGTGGAGTAAGTTCAATCATGTTAATTTCACAGAAACCG
ACAGGGAAAATAGAAGGAGAGGTTGCATGGTAACAGAACCAGTCAGATCCGTCTGCTGCTTCTGAGCCATCGATCCCAATCAT
CAGGAATCCGTCAGCTAGCACCTTTCTAATGGTTGCGACACATATTGTAGAAAGATTTAA
>bcc_1|Cycle_0|Type_1|lower_path_length_78|C1_2|C2_2|C3_14|C4_15|rank_0.55963
TGAGGTAGTCAAACCATTTAAAAGGAAGTTTTGTGTAACCTTTCTAATGGTTGCGACACATATTGTAGAAAGATTTAA
```

Figure 2: Content of the file `results_mock1_mock2_virus1_virus2_k41_coherents_type_1.fa`, when using –counts 0 option. In red are the flanking k-mers (i.e. extremities of the flanking exons). In white is the variable part (i.e. the skipped exon).

In the example, the longer splice variant is supported by 47 reads in mock1, 78 reads in mock2, 18 reads in virus1, 13 reads in virus 2. The shorter splice variant is supported by 2 reads in mock1, 3 reads in mock 2, 14 reads in virus1 and 15 reads in virus2. Overall, this exon is more often skipped in samples infected with IAV.

When using the –counts 2 option (default option in Kissplice-2.5.x), (Figure 1), the counts of the longer variant are split in 4 categories: reads fully contained in the variable part (S counts), reads spanning the junction with the upstream exon (AS counts), reads spanning the junction with the downstream exon (SB counts), reads spanning both junctions (ASSB counts). For this example, out of the 47 reads supporting the longer variant in mock 1: 15 support the first junction, 23 the second junction, 9 are included in the skipped exon and none support both junctions.

3.2 Common TroubleShooting

Before running `KisSplice` on a large dataset, (10 conditions with 100M reads each), which takes time and memory, we advise to first run `KisSplice` on a subset of your dataset (2 conditions with 10M reads each) and get familiar with the output.

If you encounter problems running `KisSplice`, this is often due to the stack size limit. Please try increasing your stack size before running `KisSplice`. The best way to do this is setting the stack size to unlimited, if your operating system accepts this, executing the following command:

```
> ulimit -s unlimited
```

If your operating system does not accept the previous command, you can increase your stack size to the highest value possible. To find this value, execute:

```
> ulimit -H -s
```

And then, to set the stack size, execute:

```
> ulimit -s <highest_value>
```

If the problem persists, please do not hesitate to contact us.

3.3 Options

Type `kissplice -h` to see the full list of options and parameters

3.4 Counts option

A read is used for quantification if and only if it overlaps (by at least `min_overlap` nt) the variable part of the path. We can distinguish various parts in the paths of an event (please see Fig. 3):

- AS: junction between the left part of the path (A in green) and the variable part of the longer path (S in red). A read is in AS if it has more than `min_overlap` nt in S and A
- SB: junction between the right part of the path (B in blue), and the variable part of the longer path (S in red). A read is in SB if it has more than `min_overlap` nt in S and B.
- S: the variable part (in red), the difference between the upper and lower path. A read is in S if it has less than `min_overlap` nt in A or B.
- AB: junction between A and B (left, resp. right part of the path) in the shorter path. A read is in AB if it has more than `min_overlap` nt in A and B.
- ASSB: the junction AS and SB with S. A read is counted in ASSB if if has more than `min_overlap` nt in A and B (and so if fully in S).



Figure 3: Example of quantification. In lower case, the context obtained using the `-output-context` parameter.

The `-counts` options can take three values: `0,1,2`. If does not affect the quantification itself, but changes the output.

By default, in kissplice-2.5.x, the counts option is set to 2. This affects only splicing events.

start=0 the total count is reported (default behaviour) as CX

stbrt=0 the counts are written for ASX, SBX, ASSBX, ABX

stcrt=0 all the counts are reported (ASX, SBX, SX, ASSBX and ABX)

Here X indicates the read file used for the quantification. It is possible to retrieve the total count C using: $C = (AS - ASSB) + (SB - ASSB) + S + ASSB$

3.5 Paired-end reads

For now, no particular treatment is given to paired-end reads. We recommend that you use them as two input files :

```
kissplice -r condition1/1 -r condition1/2
```

4 I/O

4.1 Input

`KisSplice` may be used either directly from one or several sets of reads, or from a bi-directed de-Bruijn graph.
In the first case, one or several fasta/fastq files containing the reads have to be provided. In the second case, the de-Bruijn graph alone has to be provided (in dot format). In the input fasta/q files, each read should be written on exactly two lines for fasta (one for the fasta identifier and one for the sequence) and four lines for fastq.
The input files should have one of the following extensions: `fq, fastq, txt` or `fasta, fa`. `KisSplice` also handles compressed reads (`.fa.gz`)

To have an example of the input files, run `KisSplice` on the test data (provided in the directory `sample_example`), consisting of two read files in fasta format. You will find the constructed de-Bruijn graph built by `KisSplice` in the `KisSplice` directory.

4.2 Output

If it was not provided, a de-Bruijn graph is built for the N fasta files (read) given as input; the following three files are created in the results directory (for $k = XX$):

- `graph_NameReadFile1_NameReadFile2_..._NameReadFileN_kXX.nodes` for the nodes of the de-Bruijn graph
- `graph_NameReadFile1_NameReadFile2_..._NameReadFileN_kXX.edges` for its edges.
- `graph_NameReadFile1_NameReadFile2_..._NameReadFileN_kXX.abundance` for the $k-mers$ count of each node.

In the node file, the 1st column is the node ID, the 2nd its (forward) sequence.

In the file describing the edges, the 1st and 2nd column are the IDs of the nodes that are connected by an edge, the 3rd column codes for the direction of the edge (FF = from the forward sequence of a node to the forward sequence of the other node, RR= from reverse to reverse, FR and RF). Note that if node A is connected with node B by an edge with direction "FF" ("FR"), node B is connected to node A by an edge directed "RR" ("FR") and vice versa.

Moreover, six fasta files for the results are created:

1. `results_NameReadFile1_NameReadFile2_..._NameReadFileN_kXX_type_0a.fa`: Single SNPs or sequencing substitution errors

2. `results_NameReadFile1_NameReadFile2_..._NameReadFileN_kXX_type_0b.fa`: Multiple SNPs or sequencing substitution errors
3. `results_NameReadFile1_NameReadFile2_..._NameReadFileN_kXX_type_1.fa`: alternative splicing events
4. `results_NameReadFile1_NameReadFile2_..._NameReadFileN_kXX_type_2.fa`: inexact tandem repeats
5. `results_NameReadFile1_NameReadFile2_..._NameReadFileN_kXX_type_3.fa`: short indels
6. `results_NameReadFile1_..._NameReadFileN_kXX_type_4 .fa`: all others, composed by a shorter path of length $> 2k$ not being a SNP

If the read files (option `-r`) are provided as input files, the fasta files are checked for read coherency, which means that each nucleotide of each sequence has to be covered by at least one read. If `KisSplice` is used with option `-g` (only the de-bruijn graph is provided), a check for read coherency is not possible due to missing information about read coverage.

Since the read coherency step may take a long time, `KisSplice` preemptively outputs these six fasta files before checking for read coherency in the directory `results_without_read_coherency` inside the results directory. This enables the user to access the results even before the read coherency module finishes.

`KisSplice` also creates a summary log file in the results directory named `kissplice_log_summary_(time)_(date)_(random_integer)` in case the user wants to view a summary of the execution. This is not meant to be a detailed output. If you need a detailed output, please run `KisSplice` in verbose mode (parameter `-v`) and redirect the output and error stream to a file.

Fasta files are organized as follows, each 4-lines groups are an event

```
> identifier_upper_path
sequence upper path
> identifier_lower_path
sequence lower path
```

The identifier for the upper path is formatted as follows :

```
>bcc_BB|Cycle_YY|Type_ZZ|upper_path_Length_UU|C1_cov1|C2_cov2[...]|CN_covN|rank_RR
```

- `bcc_BB` : Bi-connected component BB the event belongs to
- `Cycle_YY` : since in each bcc, several cycles may exist, this attribute indicates the ID of the cycle (here: cycle YY) that generated the bubble
- `Type_ZZ`: with `ZZ={0a, 0b, 1, 2, 3, 4}`, the type also corresponds to the type given by the file name
- `upper_path_Length_UU` : length (in nucleotides) of the sequence of the upper path of the bubble

- `Cn`: with `n={1,..., N}`: coverage of the path using reads from the read file n; coverage is the raw count of reads mapping the path with at least `k` (i.e. the value specified for the $k-mer$ length) nucleotides.

- `rank_RR` : This piece of information used to give an indication of if the alleles/the variants were differentially expressed between two conditions. It was an indication of the strength of the association between the allele/the variant and a condition but this was not a statistical test. We now provide a reliable indicator based on statistical analysis in our package kissDE (see `http://kissplice.prabi.fr/tools/kissDE/`).

The identifier for the lower path provides virtually the same information, but concerning the lower (shorter) path of the bubble. In order to facilitate the further post-processing of these files, the bcc, cycle, type and rank are repeated, although this information is redundant.

If `KisSplice` is run with option `-r` (read files), the events in the output files are sorted with respect to their rank (this is not possible with option -g due to missing information about read coverage).

4.3 Uncoherent Bubbles

In order for a bubble to be considered as coherent, each nt has to be covered by at least one read. Uncoherent bubbles are output in a separate file. In principle, uncoherent bubbles should correspond to artefacts of the DBG (we lose information when we move from reads to $k-mers$). In practice, real events which have a low coverage may produce uncoherent bubbles. Hence, it may be worth to mine this file if you are interested in unfrequent events.

5 Performances

5.1 Pre-treatment

`KisSplice` can be run on raw data. However, as any assembler, it performs better if the data is pre-treated. Common pre-treatment includes polyA tail removal and quality filter. These filters are not included in `KisSplice` distribution but can easily be found. For instance, FASTX Toolkit can be used to trim the reads, i.e. removing the last bases for which quality is below a threshold (we commonly use 20). Reads shorter than 20 nt are then removed.

5.2 Job calibration

The first time you run `KisSplice` , you may be interested in getting quickly initial results. For moderate size datasets (<500M reads), this can be achieved with default parameters. For large datasets (>500M reads), this may require to modify some parameters.

By default c is set to 1. This means that only k-mers seen more than once are considered for the analysis. k-mers seen only once are indeed likely to correspond to sequencing errors. They can also correspond to rare variants. Increasing c to 5 will correspond to a major speed-up of `KisSplice` , at the expense of a loss in sensitivity: variants supported by less than 5 reads will not be reported anymore.

By default k is set to 41. This means that repeats smaller than 41nt will not cause trouble. This also means that reads have to overlap by at least 40nt to be assembled. Increasing k to 51 will enable to solve more complex regions due to repeats, and hence will speed-up `KisSplice` , at the expense of a loss in sensitivity: reads will need to overlap by at least 50nt to be assembled.

You might be wondering about the disk space you need to have available for a job. `KisSplice` writes in several files and temporary files, and we currently consider that you must have two to three times the size of your data in free space to run `KisSplice` under proper conditions. To resolve disk space issues, we recommand to use `-d` option followed by a directory able to contain between two and three times the size of the input data.

If you want to use parallelisation option in your job, our advice is to put `-t` at 4 or 6 processors, more is not really efficient.

5.3 Relative coverage

The option `-C` edits the De-Bruijn graph constructed by removing edges non covered. It is a local filter. For a specific node, if one of its outgoing edges is covered less than `C` (in percentage) it is removed. For a small value of `C`, this option allows to remove artificial edges due to overlapping of two $k-mers$ (not supported by the reads) or sequencing errors. Increasing `-C` will reduce the run time as there will be less things to enumerate, but the price is to lose some events with a rare isoform.

5.4 Big datasets

For some datasets, enumerating all bubbles is very long. Hence, we set a maximum amount of time (100000s by default) that the algorithm spends in each BCC (biconnected component). If after this time, the BCC is not finished, the algorithm stops and moves to the next BCC. Running `KisSplice` with `-u` option enables to output these unfinished BCCs for further inspection (for instance using Cytoscape).

If you intend to work on rather big datasets ($>$ 1G reads), have a look at your `.e` log file in the end of the run. If you read `"Timeout reached"`, it means the maximum amount of time has been reached, some bccs may have been lost then. We recommand to run again KisSplice with a higher `-C` (`-C` 0.05 is a good input) to simplify the graph (while not losing too much information as there are already many reads) and retrieve all events, or to increase the timeout (option `-timeout`).

6 Further questions

Please visit our at FAQ `http://kissplice.prabi.fr/FAQ/`.