Splicing Analysis and Quantification Pipeline

Dmitri D. Pervouchine

March 19, 2014

Contents

1 Overview

This document contains a full description of main processing steps of splicing analysis and quantification pipelines. There are at least two such independent

pipelines, abbreviated here as SJPIPE and TXPIPE.

The SJPIPE pipeline implements the quantification of splicing events by directly analyzing the alignments in BAM files (Figure 1). The advantage of this approach is that it allows to quantify the annotated as well as novel splice junctions, and there is no annotation-related interference between independent splicing events. An obvious disadvantage, however, is that only a part of the sequencing information is used, one which is related to local short read distribution at splice junctions, while the reads that are contained in exons or introns are completely ignored.

In SJPIPE pipeline, the BAM file is read by *sjcount* utility to produce a tab delimited output of splice junction counts with offsets and, additionally, the respective counts of the continuous reads that overlap splice junctions (explained below). This output is aggregated (*aggregate.pl*), matched against genome and annotation (*annotate.pl*), and passed to strand disambiguation (*choose_strand.pl*). The resulting stranded counts are then subject to the irreproducibility assessment (*npIDR*) and filtering (*filter*). Importantly, this pipeline is designed to work uniformly with

- stranded and unstranded sequencing protocols
- data with and without bioreplicates

The TXPIPE pipeline implements the quantification of the annotated splicing events by analyzing the transcript quantification data produced by some other program. The advantage of this approach is that it deconvolves exon abundancies by taking all sequencing data, i.e., not only the local short read distribution at splice junctions, but also the distribution of short reads in exons and introns. This leads to a larger effective sample size compared to that in the SJPIPE pipeline. The disadvantage of this, however, is that an unannotated splicing event or a number of annotated dependent splicing events may distort the global splicing pattern significantly and, therefore, there is no guarantee that the inclusion rates of each given exon reflect actual splicing events that occur with the exon.

2 Detailed description of the components

2.1 Pipeline generator

The *make.pl* utility takes as an input an index file and outputs to the standard output a makefile to be executed to run the pipeline.

```
make.pl -dir <dirname> -param <params> -by <attribute> -margin
          <length> -annot <gtf> -genome <name> -merge <name>
```

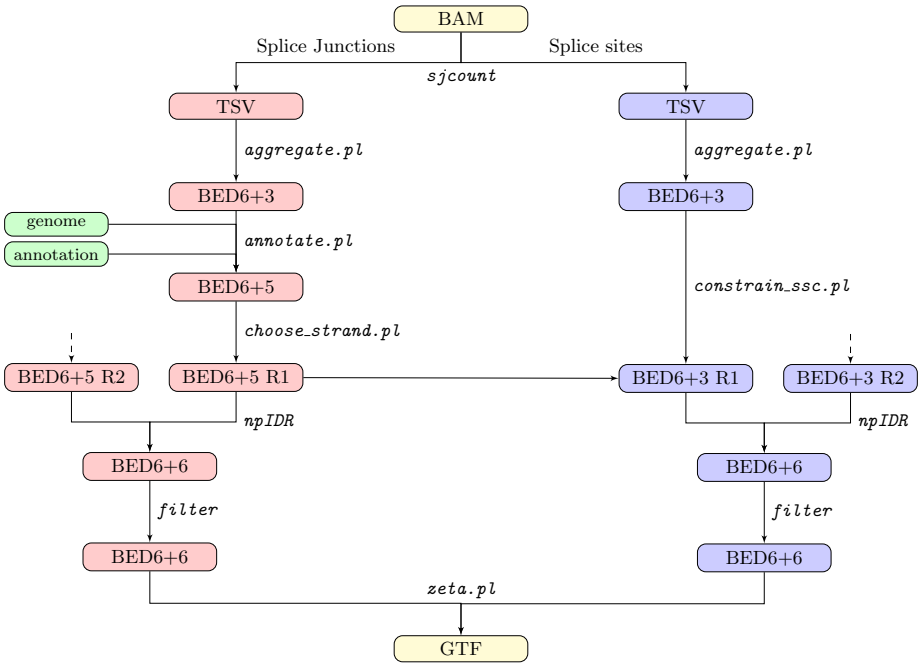


Figure 1: The SJPIPE pipeline.

```
Inputs: -annot ..., the annotation (gtf), obligatory
-block ..., the blocking field for merge
-dir ..., the output directory, obligatory
-entropy ..., entropy lower threshold, default=3
-genome ..., the genome (without .dbx or .idx), obligatory
-group ..., the grouping field for IDR, obligatory
-idr ..., IDR upper threshold, default=0.1
-margin ..., margin for aggregate, default=5
-merge ..., the name of the output to merge in case if blocks are missing
-mincount ..., min number of counts for the denominator, default=20
-param ..., parameters passed to sjcount
STDIN:  an index file
Output: STDOUT: a GNU MAKE makefile
make -f file.mk all
```

For example, the command

```
make.pl -dir data/human/ -param '-read1 0 -read2 0' -margin 4
          -annot hg19v10.gtf -genome genome/homSap19
          -group labExpId -block rnaExtract < index.txt > pipeline.mk
```

creates a file 'pipeline.mk' based on the input from 'index.txt', where all the intermediate steps will be stored in 'data/human/' (needs to created beforehand), with the annotation file hg19v10.gtf in the current directory, and with genome/homSap19.idx and genome/homSap19.dbx both readable files. The master tables will be named data/human/output*.tsv

2.2 Pre-processing

2.2.1 Annotation pre-processing

Genomic annotation files usually come in gtf format and contain many feature types. Often these are large files and it takes time to read such a file. Sometimes they may be incomplete (e.g. contain exons, but not introns). The *transcript_elements.pl* utility reads such a gtf, extracts only exons and transcript identifiers, to which these exons belong, and outputs a shorter form of gtf which contains (1) exons and (2) introns calculated based on the exon information. The format of the output is also gtf, where in the last column (column 9), the 'belongsto' attribute lists the transcripts to which given exon or intron belongs. For example, in the input gtf were

```
...   ...     ...  ... ... . . . ...
chr2L FlyBase exon 100 200 . + . gene_id "8"; transcript_id "1";
chr2L FlyBase exon 300 400 . + . gene_id "8"; transcript_id "1";
chr2L FlyBase exon 500 600 . + . gene_id "8"; transcript_id "1";
chr2L FlyBase exon 100 200 . + . gene_id "8"; transcript_id "2";
chr2L FlyBase exon 500 600 . + . gene_id "8"; transcript_id "2";
...   ...     ...  ... ... . . . ...
```

then the pre-processed, shorter form would have been

```
chr2L SJPIPE  exon    100   200   .    +    belongsto "1,2";
chr2L SJPIPE  intron  200   300   .    +    belongsto "1";
chr2L SJPIPE  intron  200   500   .    +    belongsto "2";
chr2L SJPIPE  exon    300   400   .    +    belongsto "1";
chr2L SJPIPE  intron  400   500   .    +    belongsto "1";
chr2L SJPIPE  exon    500   600   .    +    belongsto "1,2";
```

Note that transcripts and introns are not shown in the input gtf. The information contained in the 'belongsto' field will be used only in TXPIPE pipeline to compute splicing indices from transcript quantification (see section 2.5).

2.2.2 Genome pre-processing

The following two utilities, *transf* and *getsegm*, which belong to the *maptools* package, are used to pre-process genomes in a more compact and readable form.

maptools can be obtained from github. The scripts in maptools/bin shall be made accessible by declaring the path to that directory.

The use of *transf* utility is as follows

```
transf  -dir genome_directory/anyfile.fa  -dbx output.dbx -idx output.idx
```

It takes all the files in *genome_directory/* and creates two output files, *output.dbx* and *output.idx*; the former storing the data and the latter storing the index table tot hat data. The format is similar to 2bit.

The *getsegm* doesn't have to run on its own. Instead, it is used in *annotate.pl* to get genomic nucleotides.

2.3 SJPIPE pipeline

2.3.1 Counting splice junctions and reads that overlap splice sites

The *sjcount* (v.2.14) utility counts the number of split reads supporting splice junctions (SJ) and continuous reads that overlap splice sites (SS) in a BAM file. Splice sites are defined by the splice junctions that are present among the alignments. The utility returns the number of counts for each combination of chromosome, begin, end, strand, and offset, where offset is defined to be the position (within the short read sequence) of the last nucleotide preceding the splice junction. For the exact definitions of SJ, SS, offset, and examples see the help page of *sjcount2* at github.

Regarding continuous reads that overlap splice sites, the current convention is that only the full match ('M' in CIGAR string) is counted as overlapping a splice site (this is needed to get the correct offset). See also *sjcount2* (beta-testing).

```
sjcount -bam <file> -ssj <file> -ssc <file> ...
Input:  a sorted BAM file with a header
Output: -ssj, splice junction counts, and -ssc, splice site counts
Options:
-read1 0/1, reverse complement read1 no/yes (default=1)
-read2 0/1, reverse complement read2 no/yes (default=0)
-nbins number of bins for offsets, (default=1), i.e., read length
-unstranded, force strand=0
-quiet, suppress verbose output

Output columns are: chr, begin, end, strand, offset, count, e.g.
chr1    100     200     -      10      25
chr1    100     200     -      11      12
...     ...     ...     ...     ...     ...
In the ssc file begin=end=position of splice site
```

NB: the coordinates are 1-based

2.3.2 Aggregating SJ counts over offsets

The *aggregate.pl* utility takes the output of *sjcount* on STDIN and performs aggregation by the 5th column (offset) using three different aggregation functions (see examples below). It outputs a BED6+3 file with three extra columns being (7) total count, (8) staggered read count, (9) entropy. The output is sent to *stdout*.

```
aggregate.pl
Input:  TSV file (ssj or ssc) on STDIN
Output: BED6+3 on STDOUT
Options:
-margin ..., the margin for offset, default=0
-maxintron ..., max intron length, default=0
-minintron ..., min intron length, default=0
-readLength ..., the read length, default=0

Columns in the output are: chr, begin, end, name, score, strand,
count, staggered, entropy
```

It is possible to exclude short reads with small overhangs on either side by using -margin and -readLength parameters. This is particularly important when such a margin was imposed during the mapping step, but in order to be comparable when counting reads that overlap splice sites one should use the same restriction. Also, it is possible to exclude SJs that are too long or too short (-minintron/-maxintron).

The aggregation functions are applied to the sample $\{x_k\}$ of counts for each combination of chromosome, begin, end, and strand vs. the offset value k. The aggregation function therefore has the general form $f(x_1, \ldots, x_n)$.

When $f(x_1, \ldots, x_n) = x_1 + \cdots + x_n$, the result coincides with the collapsed (total) number of counts, i.e., as if offsets were ignored. For for $f(x_1, \ldots, x_n) = \theta(x_1) + \cdots + \theta(x_n)$, where $\theta(x) = 1$ for $x > 0$ and $\theta(x) = 0$ for $x \leq 0$, the result is the number of *staggered* read counts. The function

$$f(x_1, \ldots, x_n) = \log_2(\sum_{i=1}^{n} x_i) - \frac{\sum_{i=1}^{n} x_i \log_2(x_i)}{\sum_{i=1}^{n} x_i}$$

gives the entropy of the distribution, which can be used later to filter out non-uniform distiburtion of read counts. The score (field number 5 of BED) is defined to be $\min\{100 * \log_2(total), 1000\}$. For example, if the input were

```
chr1     50     90     +      20     5
chr1    100    200     -      10     25
chr1    100    200     -      11     12
chr1    100    200     -      15     4
chr1    100    200     +      10     1
chr1    100    300     +      11     12
```

the output would have been

```
...     ...     ...     ...     ...     ...     ...     ...     ...
chr1    100     200     .       536     -       41      3       1.28
...     ...     ...     ...     ...     ...     ...     ...     ...
```

where $536 = 100 * \log_2(41)$ and 1.28 is the entropy of the distribution.

Note: coordinated in BED are currently 1-based.

2.3.3 Checking the annotation status of a SJ and retrieving splice site nucleotides

The *annotate.pl* takes an aggregated BED6+3 file (i.e., the output of *aggregate.pl*) on STDIN, the genomic annotation, and the genome, and outputs BED6+5 with two more columns: (10) the annotation status and (11) splice sites. The output is sent to *stdout*.

```
annotate.pl
Input:  BED file (the output of aggregate.pl) on STDIN
-annot ..., the annotation (gff), obligatory
        -dbx ..., the genome (dbx), obligatory
        -idx ..., the genome (idx), obligatory
Input:  BED file (the output of aggregate.pl)
```

The annotation file is a simplified, processed form of the standard annotation gtf. It can be obtained by the *transcript_elements.pl* utility (see section 2.2.1). The genome consists of two compressed files, *.dbx and *.idx, which can be obtained from the genomic fasta sequence by using *transf* utility of the *maptools* package.

The annotation status is defined numerically as follows:

0 None of the splice sites of the given SJ is annotated;

1 One of the splice sites of the given SJ is annotated, and the other is not;

2 Both splice sites of the given SJ are annotated but the intron between them is not;

3 Both splice sites of the given SJ are annotated, and so is the intron between them.

The splice site nucleotides are the four intronic nucleotides, two flanking ones from each end, such as GTAG or ATAC. Since for this field and for the annotation status strand has to be defined, two lines are produced in the case of unstranded data (one for each strand). For instance, if the input were

```
...    ...   ...   ...   ...    ...   ...   ...   ...
chr1   100   200   .     536    +     41    3     1.28
chr1   100   200   .     536    -     41    3     1.28
...    ...   ...   ...   ...    ...   ...   ...   ...
```

and there were, indeed, an annotated junction at (chr1, 100, 200, –) with GTAG, then the output would have been

```
...    ...   ...   ...   ...    ...   ...   ...   ...    ...   ...
chr1   100   200   .     536    +     41    3     1.28   0     CTAC
chr1   100   200   .     536    -     41    3     1.28   3     GTAG
...    ...   ...   ...   ...    ...   ...   ...   ...    ...   ...
```

Note that sequence retriever uses the *getsegm* program of the *maptools* package, so maptools has to be installed and path has to be added.

2.3.4 Strand choice

At this step, a unique value of strand is chosen for each SJ. This is done by *choose_strand.pl* utility.

```
choose_strand.pl
Input:  BED6+3+2 file on STDIN
Output: BED6+3+2 file on STDOUT
Options:
-annot ..., annotation column, default=10
-sites ..., splice site column, default=11
```

For each combination of chromosome, begin, and end, the strand with greater annotation status (see section 2.3.3) is chosen. In case of a tie (usually 0 on both strands), the strand is chosen based on the "largest" splice site nucleotides in terms of lexicographic order (TTTT>GTAG> ...). There will be an option to choose a custom order of trustable splice site sequences (e.g., GTAG>ATAC>others).

For instance, if the input were

```
...    ...   ...   ...   ...    ...   ...   ...   ...    ...   ...
chr1   100   200   .     536    +     41    3     1.28   0     CTAC
chr1   100   200   .     536    -     41    3     1.28   3     GTAG
chr1   150   200   .     439    +     21    2     1.01   0     GTAG
```

```
chr1   150   200   .     439    -     21    2     1.01   0     CTAC
...    ...   ...   ...   ...    ...   ...   ...   ...    ...   ...
```

the output would have been

```
...    ...   ...   ...   ...    ...   ...   ...   ...    ...   ...
chr1   100   200   .     536    -     41    3     1.28   3     GTAG
chr1   150   200   .     439    +     21    2     1.01   0     GTAG
...    ...   ...   ...   ...    ...   ...   ...   ...    ...   ...
```

2.3.5 Constraining splice site counts

Since now a unique value of strand is chosen for each SJ, the counts of reads overlapping splice sites have to be constrained to a smaller set of splice sites. This is done by *constrain_ssc.pl* utility.

```
constrain_ssc.pl
Input:  the counts of reads overlapping splice sites on STDIN
-ssj ..., input ssj (bed) file, obligatory
Output: STDOUT
```

Here -ssj is the BED file after strand choice was made, -ssc is the output of *sjcount*. The output of *constrain_ssc.pl*is sent to *stdout*. If the ssc input is unstranded, then the strand of a splice site is taken from ssj, where the strand is already defined. In some cases it will lead to two lines being produced in the case of unstranded data (one for each strand). For example, if the ssj input were

```
...    ...   ...   ...   ...    ...   ...   ...   ...    ...   ...
chr1   100   200   .     536    -     41    3     1.28   3     GTAG
chr1   150   200   .     439    +     21    2     1.01   0     GTAG
...    ...   ...   ...   ...    ...   ...   ...   ...    ...   ...
```

then (chr1, 200, +) and (chr1, 200, –) are both valid splice sites and the corresponding ssc counts will be reported for each of the two strands.

2.4 Ascertainment of reproducibility (IDR)

In this step the number of counts from (generally, as many as possible) bioreplicates are assessed for irreproducibility. This step is done by *idr4sj.r*

```
Rscript R/idr4sj.r inp1.bed [inp2.bed] ... [inpN.bed] output.bed
```

where inp1,2,...N the bioreplicates and output is the file is the last in the command line. The output contains one extra column (12) equal to IDR score. Columns 7, 8, and 9 are summed (not averaged!) between bioreplicates. In case if only one input file, the IDR score is set to 0. Currently, in case of more than two bioreplicates, only the first two files will be considered (others ignored).

2.4.1 Filtering

There is no specific routine for filtering because it can be done by *awk* by requiring column 9 (entropy) to be greater than threshold (usually, 3 bits) and the column 12 (IDR) be not greater than 0.1.

2.4.2 Calculation of splicing indices

As soon as the SJ (and splice site) counts were assessed for reproducibility and filtered, the next step is to compute the inclusion and processing rates by *zeta.pl* utility. The inclusion and processing rates can be defined for exons and for introns and exist under different names [1]. Since, by definition, splice junctions know nothing about the set of exons that one might want to assess, the global exon inclusion and processing rates are computed for for a given set of annotated exons, as specified in the annotation file. In contrast, the inclusion and processing rates of SJ are computed for all splice junctions that remain intact after filtering, but also the annotated SJ are also assessed and reported.

```
zeta.pl -ssj <file> -ssc <file> -annot <gtf>
Inputs: -ssj and -ssc = SJ and SS counts in column 7;
-annot, the annotation (gtf) with exons and introns
Options:
-mincount = min denominator (will produce NA
         if denominator is smaller than this value)
-stranded = 1(yes) or 0(no), default=1
```

Here, whenever a ratio is calculated, we usually relate inclusion quantity to the sum of inclusion and exclusion quantities. The latter, however, can be a small integer number and, therefore, a threshold is needed to cut off estimates with large standard errors. This threshold is -mincount. There is also an option to enforce strandless computation, but it will be deprecated in future versions.

The procedure of *zeta.pl* is to read and to index all SJs and then for each splice site to create a list of exons which start or end at the given splice site. Then, reading sequentially the count file, the program increments the counters for exon inclusion, exon exclusion, and also for SJ usage. The output is a GFF with the corresponding features, e.g.

```
chr1 SJPIPE exon   15 87 10 - . cosi "0.93962"; exc "0"; inc "747";
                                psi "1"; ret "48";
chr1 SJPIPE intron 65 67 83 - . cosi3 "1"; cosi5 "1"; nA "0"; nD "0";
                                nDA "22"; nDX "232"; nXA "253";
                                psi3 "0.08"; psi5 "0.08661";
```

where psi and cosi are exon percent-spliced-in and completeness of splicing rates; psi5, psi3, cosi5, cosi3 are the respective percent-spliced-in and completeness of

splicing indices of an intron, measured from the 5'-end and from the 3'-end. The rest of the parameters are counts.

2.5 TXPIPE pipeline

2.5.1 Computation of splicing indices from transcript quantification data

The *tx.pl* utility takes the transcript quantification data (gtf) and the pre-processed genomic annotation (the output of *transcript_elements.pl*). For each exon in the annotation it returns the sum of abundances of transcripts that contain the given exon (as an exon, i.e., there is a line in the gtf file saying that the exon belongs to the transcript) as a fraction of the sum of abundances of transcripts which cover the given exon (i.e., the transcript starts upstream and ends downstream of the exon). Note that this definition doesn't require gene id at all.

```
tx.pl -annot <file> -quant <file>
-annot, the genomic annotation (gtf)
-quant, the transcript quantification file (gtf/gff)
```

The transcript abundance is defined in the gtf field 9 as 'RPKM' attribute or as the mean of the two bioreplicates defined by 'RPKM1' and 'RPKM2' attributes.

2.6 Master tables and endpoints

The *merge_gff.pl* utility is formally not a part of any pipeline but it can be used to merge the content of a number of gff/gtf files into a square (R-readable) matrix.

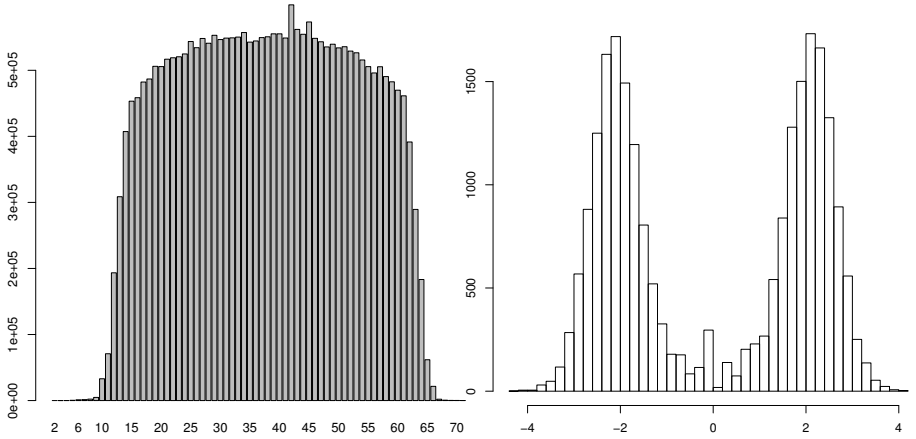
```
merge_gff.pl -i <input_file1> <label_1> ... -i <input_file_n> <label_n>
             -o feature_1 <output_1> -o feature_2 <output_2> ...
```

The program reads input files specified in the '-i' paramemter (could be many) one by one and selects features specified in '-o' paramemter. For instance,

```
merge_gff.pl -i cell_line1.gtf HELAS3 -i cell_line2.gtf NHEK
             -o psi result_psi.tsv -o cosi result_cosi.tsv
```

will generate two files, result_psi.tsv and result_cosi.tsv, each containing a square table, e.g.,

```
HELAS3    NHEK
chr1_100_200_+    0.52    0.75
chr1_300_400_+    0.00    1.00
...               ...     ...
```



(a) Offset distribution (b) Disproportion test

It can be applied to any feature that was specified in the gtf input in cell_line1.gtf and cell_line2.gtf.

3 Quality control functions

3.1 Distribution of offsets

The following utility plot the distribution of offset frequencies, i.e., how frequently each offset value was seen. This distribution may be helpful in guessing the correct margin value because some mappers have intrinsic thresholds that may be different for reads overlapping SJ and SB.

```
Rscript offset.r <file.tsv> <file.pdf>
```

An example such diagram is shown in Figure 2a.

3.2 Strand disproportion

Another useful test is the distribution of $\log(c_+) - \log(c_-)$, where c_+ and c_- are the number of counts on the plus and the minus strand, respectively. SJ with $c_+ = 0$ or $c_- = 0$ are excluded. If the data is stranded and read1/read2 flags were set up correctly, then the distribution of $\log(c_+) - \log(c_-)$ shall be bimodal as shown in Figure 2b, reflecting the fact that one strand has much more split reads than the other.

```
Rscript disproportion.r <file.bed> <file.pdf>
```

3.3 Summary stats on annotation status and splice sites

As soon as splice junction counts are computed, it makes sense to ask what proportion of splice junctions is annotated and what is the distribution of frequencies of splice site nucleotides. This is done by *sjstat.r* utility.

```
R/sjstat.r <file.bed>  >  <file.log>
```

References

[1] D. D. Pervouchine, D. G. Knowles, and R. Guigo. Intron-centric estimation of alternative splicing from RNA-seq data. *Bioinformatics*, 29(2):273–274, Jan 2013.