Quantification of Splice Junctions in RNA-seq Data by sjcount v3.0

Dmitri D. Pervouchine*

Centre for Genomic Regulation (CRG), Barcelona, Spain

September 26, 2014

Contents

1	Synopsis	1				
2	Changes since previous version					
3	Installation and usage	2				
4	1 Method					
		4				
	4.2 Offset-specific counts	5				
	4.3 Aggregation	6				
	4.4 Reads overlapping exon boundaries	6				
5	Benchmark	7				

1 Synopsis

The purpose of sjcount is to provide a fast utility for counting splice junctions in BAM files. It is the annotation-agnostic version of bam2ssj. This document describes the version $\mathbf{v3.0}$ of sjcount. The older versions of sjcount (v1.0, v2.0) is also included in the package, but their use is deprecated.

^{*}email: dp@crg.eu

2 Changes since previous version

The has been a substantial change between v2.0 and v3.0.

- 1. The utility now counts and reports all reads with multi-splits (see definitions below)
- 2. Accordingly, the output format has changed to accomidate multi-splits. One of the columns reports the number of splits (1 for splice junctions) and another column reports the coordinates of the multi-split (separated by "_").
- 3. A simpler and more efficient data structure is now used to store and parse multi-splits.
- 4. Test routines are now added to check the quality and integrity of the output as compared to the output of a pearl script which does the same counting job but has much easier syntax

3 Installation and usage

See README.md file for installation instructions. The program *sjcount* is used from the command line with the following keys

where

- bam_file is a sorted input BAM file with a header
- junctions_output is the output file with junction counts
- boundary_output is the output file with boundary counts
- read1 0/1, reverse complement read1 no/yes (default=yes)
- read2 0/1, reverse complement read2 no/yes (default=no)
- unstranded, force strand=0
- continuous, no mismatches when counting overlaps of splice boundaries
- gz, gzip output ('.gz' extension will be added to output file names)

- **nbins** number of offset bins, (default=1)
- maxnh the max value of the NH tag, (default=none)
- lim stop after reading these many lines, (default=no limit)
- quiet suppress verbose output NOTE: use -quiet if you redirect stderr to a file!

The output consists of two files. First, a tab-delimited file containing multisplit counts is produced as follows

chr1_100_200_+	1	34	1
chr1_100_200_+	1	36	1
chr1_100_200_+	1	37	6
chr1_100_200_+	1	38	3
chr1_100_200_300_400_+	2	49	1
chr1_100_400_+	1	33	1

where the first column contains the coordinates of the splits in the alignment (including multi-splits, see below). The second column contains the number of splits. The third column contains the *offset* defined as the distance within the short read sequence of the latest split (defined below). The last column is the respective number of counts, i.e., the number of split-reads with the combination of coordinates and offset specified in columns 1 and 3.

For instance, chr1_100_200_+ denotes an alignment that was split once between positions 100 and 200 on the '+' strand (referred to as 1-split), while chr1_100_200_300_400_+ denotes an alignment that was split twice (referred to as 2-split), first between positions 100 and 200, and then between positions 300 and 400. There are certain conventions on how many times 1-splits and 2-splits are counted (see below). The coordinates are always 1-based and always refer to terminal exonic nucleotides. The rationale behind it is that in applications we often need to find an intron adjacent to the given exon, and adding/subtracting one nucleotide depending on the strand becomes a headache each time. The strand is denoted by '+' and '-' for stranded data or by '.' for unstranded data.

The second output is also a tab-delimited file which contains the counts of read alignments that *overlap* exon boundaries (exon boundaries are the positions of splits, as defined above). All alignments that overlap an exon boundary by at least one nucleotide are counted (although see -continuous flag below). This second file is optional and is needed to compute the completeness of splicing index [?, ?].

4 Method

4.1 Definitions

By definition, we say that we observe a *splice junction* each time we see an 'N' symbol in the CIGAR attribute of the alignment. If the CIGAR attribute contains several N's, then we have a *multi-split* or n-split, where n is the number of N's in CIGAR. In this terms, each 1-split defines one splice junction while each n-split defines n splice junctions.

Each multi-split is counted according to the number of splits so that, for example, the alignment chr1_100_200_300_400_500_600_+ is counted once as a 3-split, two times as a 2-split (chr1_100_200_300_400_+ and chr1_300_400_500_600_+), and three times as single-split (chr1_100_200_+, chr_300_400_+, and chr1_500_600_+). The rationale behind it is that the user must be able to use awk to select n-split of interest. Therefore, all the components of, for example, a 2-split also have to be counted as 1-splits.

Note that the positions of splits are decided entirely by the mapper which produced the alignment. The mission of sjcount is to count (i) the abundancies of *n*-splits and (ii) the abundancies of reads that overlap split positions in *n*-splits. Nothing else.

As an example, consider the multi-split alignment shown in Figure 1 below. In the output file it will be counted in three lines: in chr1_31_52_+ as having 1

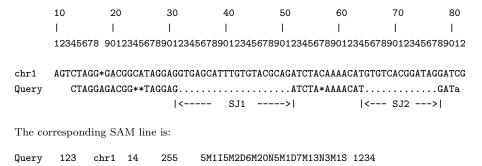


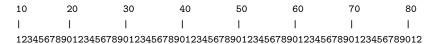
Figure 1: An example alignment and its CIGAR attribute

split, in chr1_64_78_+ as having 1 split, and in chr1_31_52_64_78_+ as having 2 splits. One may want to subset the output to regular splice junctions by requiring the second column be equal to one.

4.2 Offset-specific counts

Artifacts may arise when combining counts that come from different starting positions of the alignment. We define the *offset* to be the distance (in the query sequence!) from the first alignment position to the corresponding 'N'. For instance, the junction SJ_1 in Figure 1 has offset 17, while the junction SJ_1 has offset 29. The offset of the multi-split is defined to be the offset of it's last N, i.e., 29 in the case shown in figure 1. Since the offset is defined as a position in the query sequence, its value cannot exceed the read length.

Some offsets may give artifactually large read counts that are usually attributed to PCR artifacts [?]. In Figure 2 we show six split reads supporting the same splice junction with offsets 14 (Q1), 12 (Q2–Q4), and 8 (Q5–Q6). Note that offsets appear decreasing when sequentially processing lines a sorted BAM file.



chr1 AGTCTAGGGACGGCATAGGAGGTGAGCATTTGTGTACGCAGATCTACAAAACATGTGTCACGGATAGGATCG

Q1	GGACGGCATAGGAGATCT
Q2	ACGGCATAGGAGATCTAC
Q3	ACGGCATAGGAGATCTAC
Q4	ACGGCATAGGAGATCTAC
Q5	CATAGGAGATCTACAAAA
Q6	CATAGGAGATCTACAAAA

Figure 2: Split-mapped reads support the same splice junction with different offsets

Offset-specific counts are generated as follows. We initialize and keep nbins separate counters for each n-split. For each instance of n-split, we increment the counter corresponding to its offset. If the offset is larger than or equal to nbins then it is set to be equal to nbins - 1.

For example, in the default settings we have nbins = 1. This means that the bin number will be 1-1=0 for all supporting reads, regardless of their offset (t=14 for Q1, t=12 for Q2–Q4, and t=8 for Q5–Q6 in Figure 2). Therefore, there is only one counter to increment, and the result will be so called "collapsed" counts. The output corresponding to Figure 2 will then be

By contrast, if we set *nbins* equal to read length, there will be a separate counter for each offset and the output corresponding to Figure 2 will be

Ref_31_52_+	1	8	2
Ref_31_52_+	1	12	3
Ref 31 52 +	1	14	1

4.3 Aggregation

One single number is usually reported for each splice junction as an endpoint. Normally, the user wants to know how many reads aligned to a certain split regardless of the offset. This number is equal to the sum of counts for the given alignment over all values of offset. In other words, the total number of counts is obtained from offset-specific counts by aggregation using the function $f(x_1, \ldots, x_n) = x_1 + \cdots + x_n$.

Offset-specific counts are quite useful when aggregating by using different functions. For example, the number of staggered counts is the result of aggregation using $f(x_1, \ldots, x_n) = \theta(x_1) + \cdots + \theta(x_n)$, where $\theta(x) = 1$ for x > 0 and $\theta(x) = 0$ for $x \le 0$. Another useful function is entropy, which is obtained from the offset-specific counts by aggregation with

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

The entropy and the number of staggered reads can be used to filter out artefactual read counts. Note that *sjcount* only reports offset-specific counts, while the aggregation is left to the user. These aggregation functions are implemented in the splicing pipeline package https://github.com/pervouchine/ipsa.

4.4 Reads overlapping exon boundaries

By definition, we say that an alignment overlaps an exon boundary, if the terminal exonic nucleotide and the following first intronic nucleotide is also aligned. For example, consider exon boundaries defined by the alignment Q1 in Figure 2. In this example, Q7 and Q8 overlap the exon boundary at position 31, while Q1, Q9, and Q10 do not (note that Q1 defines exon boundary, but it would be incorrect to count it as contributing to intron retention).

```
10
          20
                 30
                        40
                               50
                                      60
                                            70
                                                   80
    {\tt AGTCTAGGGACGGCATAGGAGGTGAGCATTTGTGTACGCAGATCTACAAAACATGTGTCACGGATAGGATCG}
chr1
Q1
         GGACGCCATAGGAG.....ATCT
07
          GACGGCATAGGAGGTGA
          ACGGCATAGGAGGTGAG
Q8
Q9
           CGGCATAGGA-GTGAGC
Q9
            GGCAT.....TGTGTCACGGA
```

Figure 3: Reads overlapping exon boundaries

5 Benchmark

In principle there is not too much to benchmark in *sjcount* because it only does the job of counting. We nevertheless test the performance of *sjcount* with respect to a naïve counting routine implemented in perl. In order to initiate benchmark, call 'make test' in *sjcount* directory. It will download a small BAM file, run *sjcount*, and prompt of the outputs are different. Offset is also defined in this case as the position of the alignned exon boundary in the query sequence. For example, the offset of Q7 in Figure 3 is equal to 13. The position of exon boundary is denoted by chr_pos_str, e.g., chr1_31_+ and chr1_52_+ as in Figure 3. The number of splis is equal to 0.