# Fast quantification of splice junctions by *sjcount*

Dmitri D. Pervouchine

February 11, 2014

## 1 Synopsis

The purpose of *sjcount* is to provide a fast method for quantification of splice junctions from BAM files. It is the annotation-agnostic version of bam2ssj.

## 2 Installation and usage

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

```
sjcount -bam bam_file [-ssj junctions_output] [-ssc boundary_output]
       [-maxlen max_intron_length] [-minlen min_intron_length]
       [-margin length] [-read1 0|1] [-read2 0|1] [-unstranded]
       [-nbins number_of_bins] [-binsize bin_size]
       [-lim number_of_lines] [-quiet]
```

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
- **maxlen** upper limit on intron length, 0 = no limit (default=0)
- **minlen** lower limit on intron length, 0 = no limit (default=0)
- **margin** length, see below, (default=0)
- **read1** 0/1, reverse complement read1 no/yes (default=no)
- **read2** 0/1, reverse complement read2 no/yes (default=no)
- **unstranded**, force strand=0

- **binsize** size of the overhang bin, (default=$\infty$)
- **nbins** number of overhang bins, (default=1)
- **lim** nreads stop after nreads, (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 splice junction counts is produced as follows

```
chr1    100     200     -1      10      25
chr1    100     200     -1      11      12
... ... ... ... ... ...
```

where the first column contains chromosome, the second and the third columns contain positions of terminal exonic nucleotides defining the splice junction, the fourth column contains strand (1 or -1 for stranded or 0 for unstranded data), the fifth column is the offset (see definition below), and the last column is the respective count, i.e., the number of splices with these properties.

The secons output is also a tab-delimited file which contains read counts of alignments which *overlap* splice sites, where the latter are defined in the previous step. Only continuous alignments (not split reads) are considered. This second file is optional and is needed to compute the completness of splicing index [**?**, **?**].

# 3 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. That is, splice junctions are decided entirely by the mapper which produced the alignment. Each splice junction is characterized by a combination of four attributes: chromosome, start, end, and strand. We keep the convention that start and end of a splice junction always refer to the terminal *exonic* nucleotides. For instance, the alignment shown in Figure 1 below corresponds to two splice junctions, denoted by $\mathrm{SJ}_1$ and $\mathrm{SJ}_2$. The coordinates of these splice junctions are $\mathrm{SJ}_1$ = Ref_31_52 and SJ 2 = Ref_64_78. Denote by $l(\mathrm{SJ})$ the length of the spliced region, i.e. $l(\mathrm{SJ}_1) = 52 - 31 - 1 = 20$ and $l(\mathrm{SJ}_2) = 78 - 64 - 1 = 13$. Note that $l(\mathrm{SJ})$ is equal to the corresponding 'N' number in the CIGAR attribute.

Each splice junctions is associated with two *overhangs*, $m_u$ and $m_d$, the number of matching nucleotides immediately upstream and downstream of the junction, respectively. The numbers $m_u$ and $m_d$ are the corresponding lengths in the preceding and in the following 'M' attribute of CIGAR. For example, in Figure 1 we have $m_u(\mathrm{SJ}_1) = 6$, $m_d(\mathrm{SJ}_1) = 5$, $m_u(\mathrm{SJ}_2) = 7$, and $m_d(\mathrm{SJ}_2) = 3$.

```
       10        20        30        40        50        60        70        80
       |         |         |         |         |         |         |         |
       12345678 90123456789012345678901234567890123456789012345678901234567890123456789012

Ref    AGTCTAGG*GACGGCATAGGAGGTGAGCATTTGTGTACGCAGATCTACAAAACATGTGTCACGGATAGGATCG
Query     CTAGGAGACGG**TAGGAG.....................ATCTA*AAAACAT.............GATa
                             |<-----   SJ1  ----->|           |<--- SJ2 --->|
```

The corresponding SAM line is:

```
Query   123   Ref   14    255    5M1I5M2D6M2ON5M1D7M13N3M1S 1234
```

Figure 1: An example alignment and its CIGAR attribute

The keys **maxlen**, **minlen**, and **margin** are used as follows. For each splice junction we require that

1. $l(\mathrm{SJ}) \geq$ **minlen** and $l(\mathrm{SJ}) \leq$ **maxlen**
2. $m_u \geq$ **margin** and $m_d \geq$ **margin**

These threshold are needed to filter out non-reliable alignments with short overhangs, standartize the overhang requirement for PSI and coSI computations [**?**, **?**], and get rid of excessively long, artifactual introns.

Additionally, artifacts may arise by combining counts that come from different starting positions of the alignment. We define *offset* $t(\mathrm{SJ})$ to be the distance (*in the query sequence!*) from the first alignment position to the corresponding 'N'. For instance, $t(\mathrm{SJ}_1) = 17$ and $t(\mathrm{SJ}_2) = 29$. Since offset is defined as a refernce in the query sequence, its values are bounded by the read length.

Some offsets may give rise to artifactually large read counts [**?**]. In Figure 2 we show six split reads supporting the same splice junction with offsets 14 (Q1), 12 (Q2–Q4), and 8 (Q5–Q6).

```
       10        20        30        40        50        60        70        80
       |         |         |         |         |         |         |         |
       12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012

Ref    AGTCTAGGGACGGCATAGGAGGTGAGCATTTGTGTACGCAGATCTACAAAACATGTGTCACGGATAGGATCG

Q1            GGACGGCATAGGAG...................ATCT
Q2              ACGGCATAGGAG...................ATCTAC
Q3              ACGGCATAGGAG...................ATCTAC
Q4              ACGGCATAGGAG...................ATCTAC
Q5                  CATAGGAG...................ATCTACAAAA
Q6                  CATAGGAG...................ATCTACAAAA
```

Figure 2: Split reads support the same splice junction with different overhangs

The quantification of abundance is done as follows. For each splice junction (pair of coordinates) we initialize and keep *nbins* separate counters. For each instance of a splice junction we increment the counter corresponding to the overhang bin defined by $d = floor(v_u/binsize)$.

For example, in the default settings we have $binsize = +\infty$. This means that $d = 0$ for all supporting reads, regardless of their overhang ($t = 14$ 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 the "collapsed" counts. The output corresponding to Figure 2 will then be

```
Ref     31      52      1       0       6
```

By contrast, to take into account the overhang information, one should set $binsize = 1$ (and also specify *nbins* because the program doesn't know the range of possible overhang values). There will be a separate counter for each offset $d$ 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
```

Note that when aggregated by offset with the aggregation function $f(x_1, \ldots, x_n) = x_1 + \cdots + x_n$, the result coincides with the collapsed number of counts; 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* counts. Also

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

gives entropy of the distribution, which can be used to filter out non-uniform distiburtion of read counts.