Fast quantification of splice junctions by *sjcount*

Dmitri D. Pervouchine

September 27, 2013

1 Synopsis

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

2 Installation and usage

See README.md file for installation instructions. *sjcount* is used with the following keys

NOTE that you need to install samtools. Samtools package Version: 0.1.18-dev (r982:313) is compartible (and likely so are oder versions).

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

where

- **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)
- **binsize** size of the overhang bin, (default=∞)
- **nbins** number of overhang bins, (default=1)
- **lim** nreads stop after nreads, (default=no limit)
- **quiet** – suppress verbose output

The input to *sjcount* is a sorted BAM file with a header. The output consists of two files. First, a tab-delimited .ssj file is produced. It contains counts of splice junctions, taking into account the strand information and also start and stop positions. Second, it produces a tab-delimited .ssc file containing counts of continuous (non-split reads) which *overlap* splice sites defined by splice junctions. The second file is optional and is used to compute the completness of splicing index [].

3 Definitions

By definition, we will say that we observe a splice junction whenever we encounter 'N' symbol in the CIGAR attribute of a SAM alignment. For instance, the alignment shown in Figure 1 below gives rise to two splice junctions, SJ_1 and SJ_2. We have a convention that coordinates of splice junctions always use

```
       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    5M1I5M2D6M20N5M1D7M13N3M1S 1234
```

Figure 1: An example alignment and its CIGAR attribute

terminal exonic nucleotides, i.e., SJ_1 is Ref_31_52 and SJ 2 is Ref_64_78. We denote the length of the intron by $l(SJ)$, i.e. $l(SJ_1) = 52 - 31 - 1 = 20$ and $l(SJ_2) = 78 - 64 - 1 = 13$. Intron length is always equal to the corresponding 'N' number in the CIGAR attribute.

With each splice junctions we associate four numbers: m_u (m_d) — the number of matching nucleotides immediately upstream (downstream) of the junction, and v_u (v_d) — the length in the reference of the aligned region, also called overhang, including M/I/D operations and located immediately upstream (downstream) of the junction. in Figure 1 we have $m_u(SJ_1) = 6$, $m_d(SJ_1) = 5$, $v_u(SJ_1) = 31 - 14 + 1 = 18$, $v_d(SJ_1) = 64 - 52 + 1 = 13$ and $m_u(SJ_2) = 7$, $m_d(SJ_2) = 3$, $v_u(SJ_2) = 64 - 52 + 1 = 13$, $v_d(SJ_2) = 80 - 78 + 1 = 3$.

For each splice junction we require that

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

In addition to the coordinates of a junction, the upstream overhang $v_u(SJ)$ is also taken into account to distinguish staggered and non-staggered reads (Figure 2).

```
       10        20        30        40        50        60        70        80
       |         |         |         |         |         |         |         |
       12345678901234567890123456789012345678901234567890123456789012345678901234567890

Ref    AGTCTAGGGACGGCATAGGAGGTGAGCATTTGTGTACGCAGATCTACAAAACATGTGTCACGGATAGGATCG

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

Figure 2: Counting convention

This is done as follows. For each instance of a splice junction we increment the corresponding counter for the bin defined by $d = floor(v_u/binsize)$. For example, in the default settings $binsize = +\infty$. Then, $d = 0$ for all supporting reads, regardless of their overhang ($v_u = 14$ for Q1, $v_u = 12$ for Q2–4, and $v_u = 8$ for Q5–6). Therefore, there is only one counter to increment, and the result will be the "collapsed" counts. The output corresponding to Figure 2 will be

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

By contrast, to count split reads taking into account the overhang information, one should set $binsize = 1$ (and specify *nbins* because the program doesn't know the range of possible offsets). There will be a separate counter for each offset and the output corresponding to Figure 2 will then look like

```
Ref     31      52      1       8       2
Ref     31      52      1       12      3
Ref     31      52      1       14      1
```