**Input files**: Ziegelbauerlab>Bioinformatics>\_RNA-Seq>\_BSJ\_bed\_files

named: “\_\_\_“.BSJ.bed

**Example input:**

NC\_009333.1 27997 109676 S405 4 +

NC\_009333.1 28180 29086 S406 1 +

NC\_009333.1 28181 29086 S407 1 +

NC\_009333.1 28273 112897 S408 1 -

NC\_009333.1 28646 29488 S409 1446 +

**Input format:**

Column1=Chromosome

Column2=JunctionEnd

Column3=JunctionStart

Column4=Name

Column5=BSJCount

Column6=Strand

**Chromosome references**: Ziegelbauer>Bioinformatics>\_RNA-Seq>\_Refs

Virus assemblies:

KT899744.txt, KT899744Comp.txt

NC\_001806.2.txt, NC\_001806.2Comp.txt

NC\_009333.1.txt, NC\_009333.1Comp.txt

Human assembly:

chr1.txt, chr1Comp.txt

chr2.txt, chr2Comp.txt

chr3.txt, chr3Comp.txt

chr4.txt, chr4Comp.txt

chr5.txt, chr5Comp.txt

chr6.txt, chr6Comp.txt

chr7.txt, chr7Comp.txt

chr8.txt, chr8Comp.txt

chr9.txt, chr9Comp.txt

chr10.txt, chr10Comp.txt

chr11.txt, chr11Comp.txt

chr12.txt, chr12Comp.txt

chr13.txt, chr13Comp.txt

chr14.txt, chr14Comp.txt

chr15.txt, chr15Comp.txt

chr16.txt, chr16Comp.txt

chr17.txt, chr17Comp.txt

chr18.txt, chr18Comp.txt

chr19.txt, chr19Comp.txt

chr20.txt, chr20Comp.txt

chr21.txt, chr21Comp.txt

chr22.txt, chr22Comp.txt

chrX.txt, chrXComp.txt

chrY.txt, chrYComp.txt

**Splice DA Ref:** SpliceDAOptions

SpliceDAOptions.txt

1. **Determine the sequence -30 to +30 of the back-splice junction (BSJ) and the sequence of the splice donor-acceptor**

**First output: “1.BSJ\_Summary”**

Deposit these files in Outputs > Final\_BSJAnalysis > 1.BSJ\_Summary

**Logic:** Use positions to pull sequence information from a reference file (see java script BSJtoNtHeatmaps.java)

**Column 7 (BSJFlankingSeq)** in the output will be generated using the following logic:

Input C1 determines which reference file to pull from, i.e. if C1=KT899744 you will use the KT899744.txt file for pulling sequences. If C1 does not match any reference files located in the reference folder than that entry/row should be skipped

If Input C6= ”+” Pull sequence information from references: KT899744.txt or NC\_001806.2.txt or NC\_009333.1.txt

If Input C6= “-“ Pull sequence information from references: KT899744Comp.txt or NC\_001806.2Comp.txt or NC\_009333.1Comp.txt

sequence that is from: -30 Input C3 to Input C3 + sequence that is from: Input C2 to Input C2 +30

**Column 8 (SpliceD-A)** in the output will be generated using the following logic:

If Input C6= ”+” Pull sequence information from references: KT899744.txt or NC\_001806.2.txt or NC\_009333.1.txt

If Input C6= “-“ Pull sequence information from references: KT899744Comp.txt or NC\_001806.2Comp.txt or NC\_009333.1Comp.txt

sequence that is from: Input C3 to Input C3 +2 + sequence that is from: Input C2 -2 to Input C2

**Example output:**

Chromosome JunctionEnd JunctionStart Name Count Strand BSJFlankingSeq SpliceD-A

NC\_009333.1 122680 123279 S405 4 + GTCTTTGGGTCAACTAAGGCTTTTGTAATCAGGGCTGAAATAACTCATTGTGCCCGCTAG 1 TACA

NC\_009333.1 123232 123463 S406 3 - GGGAAAAGGAGTCTGCCGCGGCATAGCAAAAGGCGGGCAATGAGCCAGTCTTTGGGTCAA CATA

NC\_009333.1 118906 136573 S407 1 + ACAAGTTCCAAAGTTTAAGGACAATGCAAGCTGGTTCCTGGGGTGTGCCAGGACGGGTTC GATA

**Output format (.csv):**

Print the following header at row 1:

Chromosome JunctionEnd JunctionStart Name BSJCount Strand BSJFlankingSeq SpliceD-A

Column1=Input C1

Column2=Input C2

Column3=Input C3

Column4=Input C4

Column5=Input C5

Column6=Input C6

Column7=Sequence +30/-30 from BSJ (Sequence from “B” file, or -30 to End Junction) + Sequence from “A” file, or Start Junction to +30

Column8=Splice Donor (Input C3 to Input C3 +2) + Splice Acceptor (Input C2 -2 to Input C2)

**Second output: “2.BSJ\_Nt\_Matrix”**

Deposit these files in Outputs > Final\_BSJAnalysis > 2.BSJ\_Nt\_Matrix

**Input files**: Ziegelbauerlab>Bioinformatics>\_RNA-Seq>Outputs>Final\_BSJAnalysis > 1.BSJ\_Summary

**Logic:** Use newly generated BSJ\_Summary files and multiply the sequence by the number of times identified in the sample, i.e. if C5 has a value of 6 then the sequence in column 7 would be multiplied by 6x, convert nucleotides to integers, such that A/a=1, T/t=2, G/g=3, C/c=4 (see java script BSJtoHeatmap)

**Example output:**

SD2\_Rep1\_CDV\_28862-33622 4 3 4 4 2 4 3 1 3 3 3 4 3 3 4 4 1 3 2 1 2 2 1 2 3 4 4 1 3 3 2 2 2 3 2 3 4 2 4 3 4 2 3 4 2 2 3 4 4 2 2 4 2 2 3 4 4 3 4 2

SD2\_Rep1\_CDV\_28843-29575 1 2 2 3 3 1 4 2 1 1 1 3 2 3 3 2 3 2 3 4 3 3 4 1 3 4 2 3 3 3 2 3 4 4 3 4 2 2 4 1 4 4 2 1 2 3 3 1 2 2 2 2 3 2 3 4 2 4 3 4

**Output format:**

C1=Input C4

C2 through C61=Multiply BSJ sequence by number of times identified in the sample, i.e. for FUSIONJUNC\_0/1 the sequence would be multiplied by 1x, convert nucleotides to integers, such that A/a=1, T/t=2, G/g=3, C/c=4 (see java script BSJtoHeatmap)

**Third output: “3.SpliceDAFrequency”**

Deposit these files in Outputs > Final\_BSJAnalysis > 3.SpliceDAFrequency

**Input files**: Ziegelbauerlab>Bioinformatics>\_RNA-Seq>Outputs>Final\_BSJAnalysis > 1.BSJ\_Summary

**Logic:** Use newly generated BSJ\_Summary files to count the number of times each C8 unique 4 character sequence was found in a sample, making sure to multiply the sequence by the number of times identified in C5. See txt file listing out all possible 4 character options in: \_Refs > “SpliceDAOptions.txt”

**Example output:**

Splice D-A AAAA AAAC AAAG…

Sample name 68 37 40

**Output format:**

Row 1: Header line

Row2: aggregated counts for sample 1

Row3: aggregated counts for sample 2…

1. **Generate Sashimi plots**

**Input files**: Ziegelbauerlab>Bioinformatics>\_RNA-Seq>\_BSJ\_bed\_files

named: “\_\_\_“.BSJ.bed

**Third output: “4.Sashimi”**

Deposit these files in Outputs > Final\_BSJAnalysis > 3.Sashimi

**Logic:** Use “\_\_\_”.BSJ.bed files to create a junction track that can be visualized in IGV as a sashimi plot

[**https://software.broadinstitute.org/software/igv/Sashimi**](https://software.broadinstitute.org/software/igv/Sashimi)

**Input format:**

Column1=Chromosome

Column2=JunctionEnd

Column3=JunctionStart

Column4=Name

Column5=BSJCount

Column6=Strand

**Logic:**

Need to use boolean tags “isHSV1” or “isKT” to only read lines in which column1 matches the appropriate column name. So for isHSV1=true; isKT=true, only lines with column 1 containing should be read and printed. For isHSV1=false, isKT=false, only lines with column 1 containing NC\_009333.1 should be read and printed.

Only print new lines if the “count” or output column 5 value is equal or greater than the threshold variable

blocksize = 1

**Variables that should be fed in from the control App:**

threshold = 1 or 5

**Example output:**

track name=junctions

KT899744 415 2496 S32 2 + 415 2496 0,0,255 2 1,1 0,2080

KT899744 554 1496 S34 3 + 554 1496 0,0,255 2 1,1 0,941

KT899744 606 106001 S38 3 + 606 106001 0,0,255 2 1,1 0,105394

KT899744 731 1163 S43 2 + 731 1163 0,0,255 2 1,1 0,431

KT899744 856 2049 S53 2 + 856 2049 0,0,255 2 1,1 0,1192

**Output format:**

Line 1 needs to be: “track name=junctions”

Column1= Chromosome Name (only KT899744 or NC\_009333.1)

Column2= Input C2-blocksize

Column3= Input C3+blocksize

Column4= Input C4

Column5= Input C5

Column6= Input C6

Column7= Input C2

Column8= Input C3

Column9= 0,0,255

Column10= 2

Column11= 1,1

Column12= 0,(Input C3-Input C2+blocksize)

**Example prior code:**

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

public class NewBsjToSashimi {

public static void execute(String bedFileDir) throws IOException {

int threshold = 1;//minimal reads requirement

int blockSize = 1;//nt

File bedDir = new File("./\_BSJ\_bed\_files/"+bedFileDir);

String[] bedList = bedDir.list();

for(String bed:bedList) {

if(bed.endsWith("BSJ.bed")) {

BufferedWriter out = new BufferedWriter(new FileWriter(new File(bedDir.getAbsolutePath()+"/"+bed+".sashimi.bed")));

out.write("track name=junctions" + "\n");

BufferedReader br = new BufferedReader(new FileReader(bedDir.getAbsolutePath()+"/"+bed));

String line = "";

//Need to add in line to parse for only viral (NC or KT)

while((line = br.readLine()) != null) {

String[] tokens = line.split("\t");

int start = Integer.parseInt(tokens[1])-blockSize;

int end = Integer.parseInt(tokens[2])+blockSize;

int blockEnd = end-start-blockSize;

int count = Integer.parseInt(tokens[4]);

String strand = tokens[5];

String name = tokens[3];

if(threshold<count)

out.write("KT899744" + "\t" + start + "\t" + end + "\t" + name + "\t" + count + "\t" + strand + "\t" + start + "\t" + end + "\t" + "0,0,255" + "\t" + "2" + "\t" + blockSize+","+blockSize + "\t" + "0,"+blockEnd + "\n");

}

br.close();

out.close();

}

}

}

}