PSSV User Manual (V2.1)

1. Introduction

A novel pattern-based probabilistic approach, PSSV, is developed to identify somatic structural variations from WGS data. Specifically, discordant and concordant read counts from paired samples are jointly modeled in a Bayesian framework. Each type of read counts at SV regions is assumed to follow a mixture of Poisson distributions with three components to denote ‘non-mutation’, ‘heterozygous’ and ‘homozygous’ SVs in tumor or normal samples. Then, each SV is modeled as a mixture of hidden states representing different somatic and germline mutation patterns. As a unique feature of this model, we can differentiate heterozygous and homozygous SVs in each sample, enabling the identification of those somatic SVs with heterozygous mutations in normal samples and homozygous mutations in tumor samples. An Expectation-maximization (EM) algorithm is used to iteratively estimate the model parameters and the posterior probability for each somatic SV region.

We present an example of the PSSV workflow using a pair of tumor and normal DNA-seq bam files (obtained from a patient sample TCGA-A2-A0D0 of the TCGA data set); more details of the workflow can be found in our paper: “PSSV: A novel pattern-based probabilistic approach for somatic structure variation identification”. The R script of PSSV has been tested using R 3.3 under MAC OS 10.11 and Ubuntu 12.04 64 bit.

2. DNA-seq data preprocessing

With the BAM format WGS data from a pair of tumor-normal samples, we use a hierarchical clustering approach (which is embedded in **BreakDancer v1.3.6** https://github.com/genome/breakdancer) to generate discordant read clusters in each sample, and filter SVs by setting that the number of discordant reads in tumor sample is no less than 4. Here, users can also control the SV length scale using –m option of the BreakDancer-Max function. We then use **GATK** (https://software.broadinstitute.org/gatk/) to calculate the average concordant read depth within each candidate SV as well as at its two flanks in both samples. The concordant read depth within mutation regions can improve the sensitivity of SV prediction in each sample, leading to improved accuracy of somatic calling. The read depth signals at two flanks can be used to normalize local read counts so as to eliminate the GC bias and make the read counts comparable across all mutation regions. A flowchart of DNA-seq data preprocessing for somatic SV detection is shown in Fig. 1. There is one step called 'Tumor sample purification', which requires the input of normal cell proportion. An estimation of TCGA tumor samples can be found from doi:10.1038/ncomms9971.

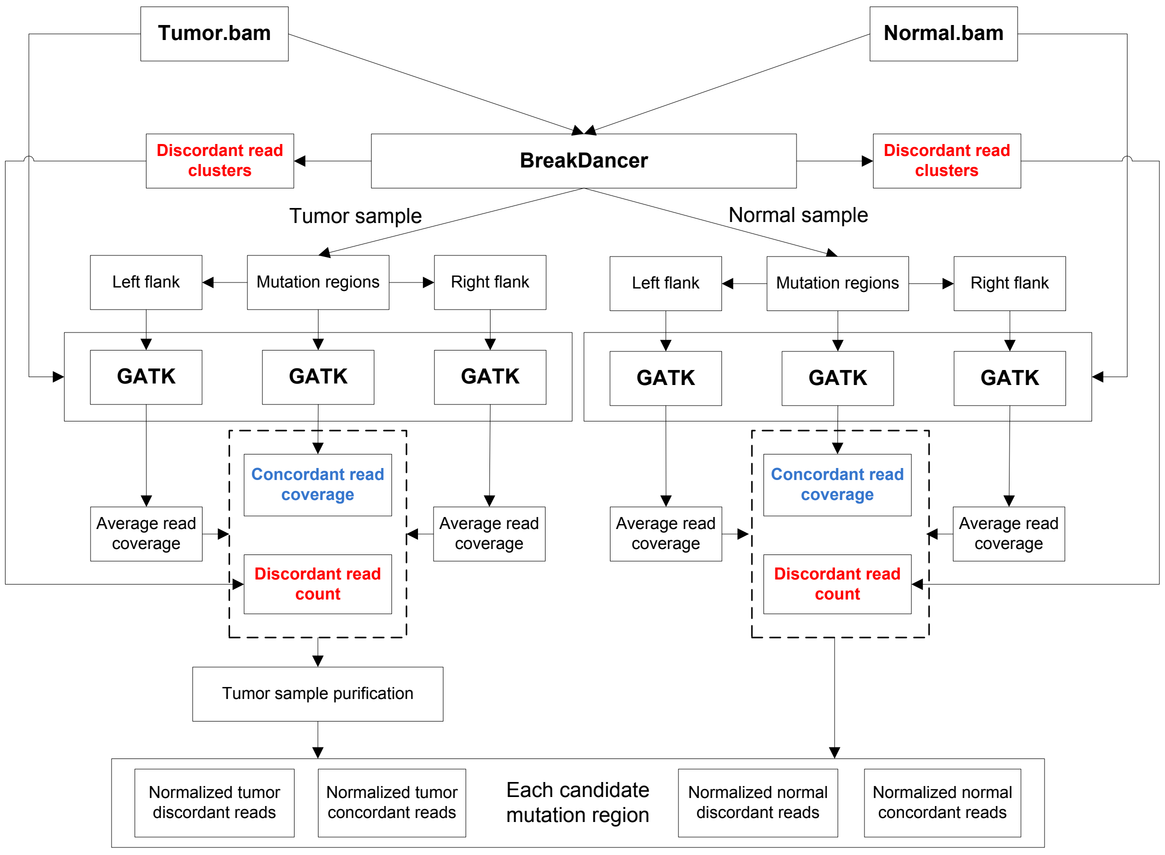

Fig. 1. Flow chart of whole genome DNA-seq data processing for somatic SV detection.

2.1 Discordant reads clustering for SV selection

We use **BreakDancer** to identify SV regions using a pair of tumor and normal whole genome DNA-seq bam files with the following command:

```
$ ./bam2cfg.pl -g -h TCGA-A2-A0D0-01A.bam TCGA-A2-A0D0-10A.bam > TCGA-A2-
A0D0.cfg
$ ./BreakDancerMax.pl -t -q 0 -f -d TCGA-A2-A0D0.cfg > TCGA-A2-A0D0.ctx
```

Note that, for each SV, BreakDancer uses the sum of discordant reads at each region observed from 'both' tumor and normal samples to evaluate its significance. However, in our case, we are more focused on the different feature between tumor and normal samples for each region. Above significance calculation cannot be used to select candidate regions to be investigated by our PSSV method. Instead, we take all possible regions based on the hierarchical clustering results of discordant reads in each sample. In the above command, we set threshold as `-q 0` to obtain all regions.

Then, to obtain the number of discordant reads at each mutation region respectively from tumor and normal samples, we have provided a small binary tool as `breakdancer_seperate_one` to convert the output of BreakDancer `TCGA-A2-A0D0.ctx` into a more concise file as `TCGA-A2-A0D0-sep.txt`.

```
$ ./breakdancer_seperate_one TCGA-A2-A0D0.ctx TCGA-A2-A0D0-sep.txt
```

The data format of `TCGA-A2-A0D0-sep.txt` is shown as follows:

```
chr1  point1      chr2  point2      type  size  score tumor normal
1     10211       1     10296       DEL   96    99    5     8
1     10306       10    135524589   INS   -161  99    4     8
1     10449       7     10001       INS   -108  58    0     3
1     16450       1     16453       INS   -196  48    2     1
1     99098       12    66451267    DEL   95    33    0     2
1     136741      1     136875      DEL   127   46    1     1
1     537692      1     537723      INS   -167  69    4     0
1     565259      1     565311      INV   -304  99    13    0
1     569310      1     569405      INS   -154  99    3     4
…
```

Now, we have a list of SVs with discordant read counts in tumor or normal samples. The discordant read clustering is time consuming. PSSV does not do read clustering by itself. Discordant read counts for deletion, insertion, inversion and translocation from a pair of tumor and normal samples are required to run PSSV.

2.2 Concordant reads coverage calculation for detected SVs

For normalization purpose, we need to estimate read coverage with each mutation region as well as its two flanks. In this package, we have provided a short R script `interval_transfer.R` to extract SV locations (mainly for deletions, insertion and inversions) as well as flank regions. Here, each flank region is defined as a 500 bps region either on the left or right side of a SV. Using `TCGA-A2-A0D0-sep.txt` as input, three files will be created by `interval_transfer.R` as `interval.bed`, `left.bed` and `right.bed`. The data format of these three files meets the requirement of GATK for read coverage estimation.

```
    interval.bed               left.bed                     right.bed

    1  10212    10296          1  9713     10212            1  10296    10795
    1  537693   537723         1  537193   537692           1  537723   538222
    1  565260   565311         1  564760   565259           1  565311   565810
    1  1034565  1034882        1  1034065  1034564          1  1034882  1035381
    1  1086842  1087027        1  1086342  1086841          1  1087027  1087526
    1  1088680  1088912        1  1088180  1088679          1  1088912  1089411
    1  1131210  1131260        1  1130710  1131209          1  1131260  1131759
    1  1598458  1598747        1  1597958  1598457          1  1598747  1599246
…
```

GATK is then used to estimate the average read coverage of each region (mutation or flank regions) using whole genome DNA-seq data of tumor and normal samples respectively.

```
$ java -jar GenomeAnalysisTK.jar -T DepthOfCoverage -R hg19.fasta -o TCGA-A2-
A0D0-interval-tumor -I TCGA-A2-A0D0-01A.bam -L interval.bed

$ java -jar GenomeAnalysisTK.jar -T DepthOfCoverage -R hg19.fasta -o TCGA-A2-
A0D0-interval-normal -I TCGA-A2-A0D0-10A.bam -L interval.bed
```

In total, six files ending with `sample_interval_summary` will be generated including read coverage information at SV regions as well as their two flanks in both tumor and normal samples.

2.3 Tumor sample impurity estimation

As reported by http://dx.doi.org/10.1038/ncomms9971, the proportion of normal cells in TCGA breast cancer TCGA-A2-A0D0 is 0.1829.

```
Theta=0.1829
```

Note: there is also some other software that can be used to solve the same problem. We do not make any judgement of the performance of different such tools. Users can select one according to their own preference.

3 Somatic SV prediction using PSSV

3.1 Discordant and concordant read information integration

After generating discordant and concordant read info using the above pipeline, we provide a demo R script as `PSSV_demo.R` to detect somatic SVs, including deletions, insertions, inversions and chromosome translocations. We first load all data to the workspace of R as follows:

```
sample_ID="TCGA-A2-A0D0"
#proportion of normal cells in the tumor sample
Theta=0.1829

#load discordant reads in tumor or normal samples
Discordant_reads<-as.matrix(read.table(paste(sample_ID,'-sep.txt', sep=''),
header = TRUE))

#load concordant read coverage at mutation regions of tumor sample
Tumor_interval<-as.matrix(read.table(paste(sample_ID,'-interval-
tumor.coverage.sample_interval_summary', sep=''), colClasses = c("character",
"NULL","character", rep("NULL", 6)),  header = TRUE))

#load concordant read coverage at left flank of tumor sample
Tumor_left<-as.matrix(read.table(paste(sample_ID,'-left-
tumor.coverage.sample_interval_summary', sep=''), colClasses = c("character",
"NULL","character", rep("NULL", 6)),  header = TRUE))

#load concordant read coverage at right flank of tumor sample
Tumor_right<-as.matrix(read.table(paste(sample_ID,'-right-
tumor.coverage.sample_interval_summary', sep=''), colClasses = c("character",
"NULL","character", rep("NULL", 6)),  header = TRUE))

#load concordant read coverage at mutation regions of normal sample
Normal_interval<-as.matrix(read.table(paste(sample_ID,'-interval-
normal.coverage.sample_interval_summary',  sep=''),  colClasses  =  c("character",
"NULL","character", rep("NULL", 6)),  header = TRUE))

#load concordant read coverage at left flank of normal sample
Normal_left<-as.matrix(read.table(paste(sample_ID,'-left-
normal.coverage.sample_interval_summary',  sep=''),  colClasses  =  c("character",
"NULL","character", rep("NULL", 6)),  header = TRUE))

#load concordant read coverage at right flank of normal sample
Normal_right<-as.matrix(read.table(paste(sample_ID,'-right-
normal.coverage.sample_interval_summary',  sep=''),  colClasses  =  c("character",
"NULL","character", rep("NULL", 6)),  header = TRUE))
```

Then, we use the following function `Discordant_concordant_integration` to integrate all data info together:

```
Candidate_regions<-Discordant_concordant_integration(Discordant_reads,
Tumor_interval_location, Tumor_left_location, Tumor_right_location,
Normal_interval_location, Normal_left_location, Normal_right_location,
Tumor_avg_Coverage, Normal_avg_Coverage)
```

The output variable `Candidate_regions` is a N x 13 matrix. Each row represents a candidate SV and each column is defined as follow:

```
Candidate_regions[,1] # chromosome ID of ‘left’ breakpoint
Candidate_regions[,2] # position of ‘left’ breakpoint
Candidate_regions[,3] # chromosome ID of ‘right’ breakpoint
Candidate_regions[,4] # position of ‘right’ breakpoint
Candidate_regions[,5] # number of discordant reads in ‘tumor’ sample
Candidate_regions[,6] # ARD within mutation region in ‘tumor’ sample
Candidate_regions[,7] # mutation type (1: DEL, 2: INS, 3: INV, 4: TRANS)
Candidate_regions[,8] # number of discordant reads in ‘normal’ sample
Candidate_regions[,9] # ARD within mutation region in ‘normal’ sample
Candidate_regions[,10] # ARD of ‘left’ flank region in ‘tumor’ sample
Candidate_regions[,11] # ARD of ‘right’ flank region in ‘tumor ‘sample
Candidate_regions[,12] # ARD of ‘left’ flank region in ‘normal’ sample
Candidate_regions[,13] # ARD of ‘right’ region in ‘normal’ sample
```

3.2 Tumor sample purification and data normalization

This is a very important step in tumor sample based genomic data analysis. We use a function `tumor_sample_purification` to achieve tumor sample purification and data normalization. Details of of this function is specifically presented here.

```
tumor_sample_purfication = function(Candidate_regions, Theta)
{
Purified_Candidate_regions=Candidate_regions
Purified_Candidate_regions[which(is.nan(Purified_Candidate_regions)=='TRUE')]=0

for (k in 1:nrow(Purified_Candidate_regions))
 {
  aa=max(Purified_Candidate_regions[k,10],Purified_Candidate_regions[k,11])
  if (aa>1)
     {
        Purified_Candidate_regions[k,5]=round(Purified_Candidate_regions[k,5]*Tumor_avg_Coverage/
        max(Purified_Candidate_regions[k,10],Purified_Candidate_regions[k,11]))
        Purified_Candidate_regions[k,6]=round(Purified_Candidate_regions[k,6]*Tumor_avg_Coverage/
        max(Purified_Candidate_regions[k,10],Purified_Candidate_regions[k,11]))
     }

  bb=max(Purified_Candidate_regions[k,12],Purified_Candidate_regions[k,13])
  if (bb>1)
     {
        Purified_Candidate_regions[k,8]=round(Purified_Candidate_regions[k,8]*Normal_avg_Coverage
        /max(Purified_Candidate_regions[k,12],Purified_Candidate_regions[k,13]))
        Purified_Candidate_regions[k,9]=round(Purified_Candidate_regions[k,9]*Normal_avg_Coverage
        /max(Purified_Candidate_regions[k,12],Purified_Candidate_regions[k,13]))
     }

  #Normalize discordant read count in tumor sample
  Purified_Candidate_regions[k,5]=round((Candidate_regions[k,5]
               -Theta*Tumor_avg_Coverage/Normal_avg_Coverage*Candidate_regions[k,8])/(1-Theta))

#Normalize read coverage in tumor sample
  Purified_Candidate_regions[k,6]=round((Candidate_regions[k,6]
               -Theta*Tumor_avg_Coverage/Normal_avg_Coverage*Candidate_regions[k,9])/(1-Theta))
 }
return(Purified_Candidate_regions[,c(1:9)])
}
```

The output format of `Purified_Candidate_regions` is similar to the first 9 columns of `Candidate_regions` but with normalized read count information.

3.3 Somatic deletion prediction

We have four different functions to detect deletions, insertions, inversions and translocations respectively. They are independent from each other so the user can select the specific SV type or types to predict.

Somatic deletion calling is achieved using function `Somatic_del_detection`.

```
DEL_results<-Somatic_del_detection(Purified_Candidate_regions,  length_control,
Tumor_avg_Coverage, Normal_avg_Coverage, L)
```

The output variable `DEL_results` contains following information:

```
DEL_results$DEL_locations[,1] #chromosome ID for deletion region
DEL_results$DEL_locations[,2] #left breakpoint
DEL_results$DEL_locations[,3] #right breakpoint

DEL_results$DEL_reads[,1] #discordant read count in the tumor sample
DEL_results$DEL_reads[,2] #read coverage in the tumor sample
DEL_results$DEL_reads[,3] #discordant read count in the normal sample
DEL_results$DEL_reads[,4] #read coverage in the normal sample

DEL_lambda # estimated Poisson mean parameters for each component
DEL_state #predicted mutation state for each deletion
DEL_probability #posterior probability for each somatic deletion
```

We present a demo histogram of mutation states of all candidate deletions of TCGA-A2-A0D0.

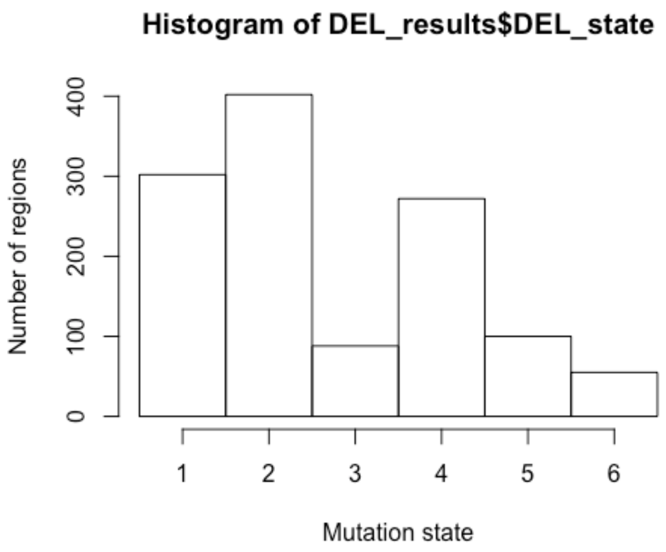

Fig. 2. Number of deletion regions assigned to each state.

3.4 Somatic insertion prediction

Somatic insertion calling is achieved using function `Somatic_ins_detection`.

```
INS_results<-Somatic_ins_detection(Purified_Candidate_regions,
Tumor_avg_Coverage, Normal_avg_Coverage, L)
```

The output variable `INS_results` contains following information:

```
INS_results$INS_locations[,1] #chromosome ID for insertion region
INS_results$INS_locations[,2] #left breakpoint
INS_results$INS_locations[,3] #right breakpoint

INS_results$INS_reads[,1] #discordant read count in the tumor sample
INS_results$INS_reads[,2] #read coverage in the tumor sample
INS_results$INS_reads[,3] #discordant read count in the normal sample
INS_results$INS_reads[,4] #read coverage in the normal sample

INS_lambda # estimated Poisson mean parameters for each component
INS_state #predicted mutation state for each insertion
INS_probability #posterior probability for each somatic insertion
```

We present a demo histogram of mutation states of all candidate insertions of TCGA-A2-A0D0.

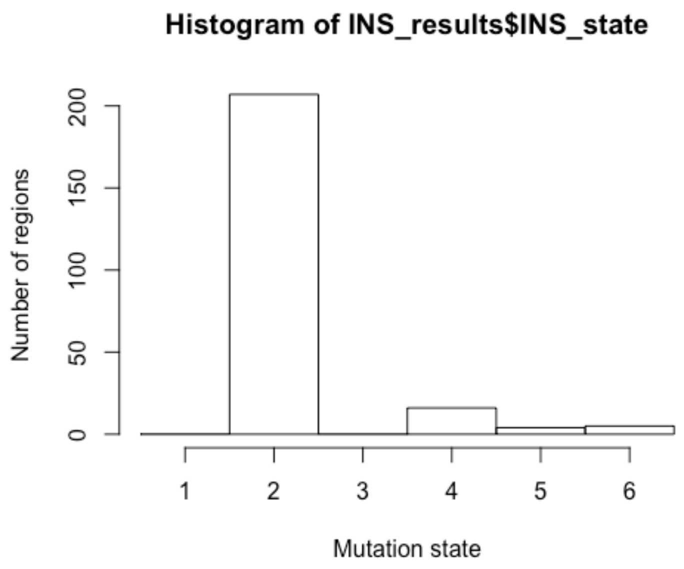

Fig. 3. Number of insertion regions assigned to each state.

3.5 Somatic inversion prediction

Somatic insertion calling is achieved using function `Somatic_inv_detection`.

```
INV_results<-Somatic_inv_detection(Purified_Candidate_regions,  length_control,
Tumor_avg_Coverage, Normal_avg_Coverage, L)
```

The output variable `INV_results` contains following information:

```
INV_results$INV_locations[,1] #chromosome ID for inversion region
INV_results$INV_locations[,2] #left breakpoint
INV_results$INV_locations[,3] #right breakpoint

INV_results$INV_reads[,1] #discordant read count in the tumor sample
INV_results$INV_reads[,2] #read coverage in the tumor sample
INV_results$INV_reads[,3] #discordant read count in the normal sample
INV_results$INV_reads[,4] #read coverage in the normal sample

INV_lambda # estimated Poisson mean parameters for each component
INV_state #predicted mutation state for each inversion
INV_probability #posterior probability for each somatic inversion
```

We present a demo histogram of mutation states of all candidate inversions of TCGA-A2-A0D0.

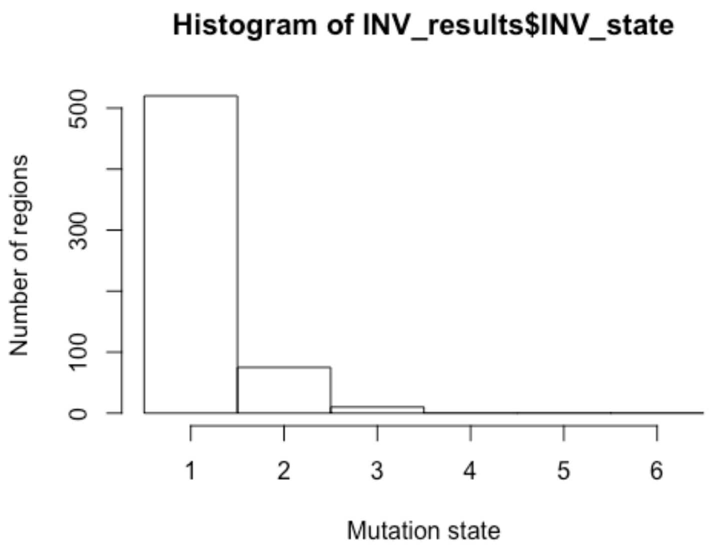

Fig. 4. Number of inversion regions assigned to each state.

3.6 Somatic inter- or intra-chromosome translocation prediction

Somatic translocations calling is achieved using function `Somatic_trans_detection`.

```
TRANS_results<-Somatic_trans_detection(Purified_Candidate_regions,
Tumor_avg_Coverage, Normal_avg_Coverage, L)
```

The output variable `TRANS_results` contains following information:

```
TRANS_results$TRANS_locations[,1] #chromosome ID for left breakpoint
TRANS_results$TRANS_locations[,2] #left breakpoint
TRANS_results$TRANS_locations[,3] #chromosome ID for right breakpoint
TRANS_results$TRANS_locations[,4] #right breakpoint

TRANS_results$TRANS_reads[,1] #discordant read count in the tumor sample
TRANS_results$TRANS_reads[,2] #read coverage in the tumor sample
TRANS_results$TRANS_reads[,3] #discordant read count in the normal sample
TRANS_results$TRANS_reads[,4] #read coverage in the normal sample

TRANS_lambda #estimated Poisson mean parameters for each component
TRANS_state #predicted mutation state for each inversion
TRANS_probability #posterior probability for each somatic inversion
```

We present a demo histogram of mutation states of all candidate inversions of TCGA-A2-A0D0.

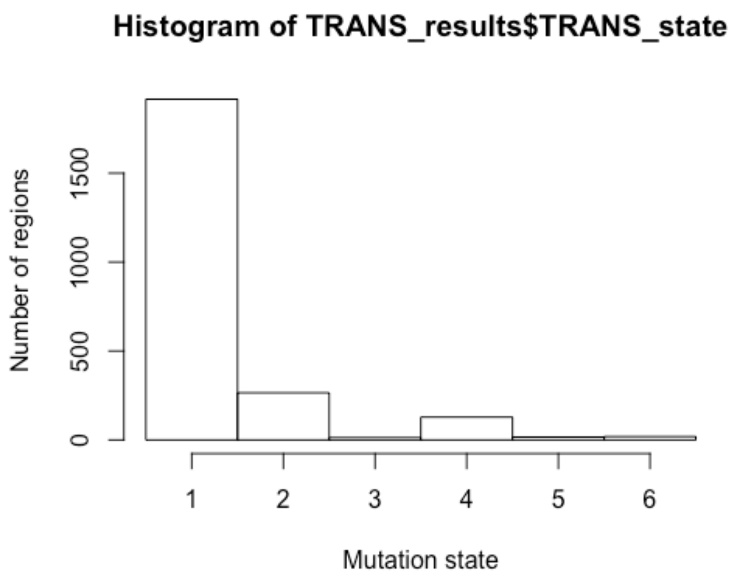

Fig. 5. Number of translocation regions assigned to each state.

4. Gene annotation (optional)

Finally, as described in our PSSV paper, we are mainly focused on those somatic SVs that lay in gene promoter and coding regions. Therefore, with gene annotation file `hg19_RefSeq.txt`, we use the following command to annotate each region with a gene if they overlap.

```
#*****************load reference genome hg19 **********************

Transcripts_hg19<-as.matrix(read.table("hg19_RefSeq.txt", header = TRUE))

#annotate each region with gene promoter (positive or negative 10k bps) and
body locations

genes_with_SVs<-annotate_SVs(Transcripts_hg19, DEL_results, INS_results,
INV_results, TRANS_results)
```

For deletions, insertions and inversions we expect that the mutation occurs at one gene's functional region. But for translocations, they may cross two different chromosome or very distant locations on the same chromosome. Hence, we do gene annotation at each breakpoint of chromosome translocations.

This gene annotation step is optional depending on whether the study is focused on SVs related to gene function or not.