

# StructuralVariantUtil

## Table of Contents

<b><i>Introduction</i></b> .....	<b>2</b>
<b><i>Installation</i></b> .....	<b>2</b>
<b><i>Third Party Software</i></b> .....	<b>2</b>
<b><i>Main Utilities</i></b> .....	<b>2</b>
SV calling performance estimation.....	2
Convert VCF format to a R data frame .....	4
Simple SV type classification .....	6
SV caller Integration filtration.....	10
Spectrum of SV types for large cohort.....	12
Spectrum of SV breakpoints in genomic bins.....	14
Copy number changes of deletion and duplication events .....	17
SV breakpoint gene annotation .....	19

## Introduction

*StructuralVariantUtil* is a R package providing utility functions for structural variant (SV) analyses, including estimating SV calling performance, parsing SV VCF, integrating results from different SV callers and cohort-level spectrum analysis. This user guide details how to install and use *StructuralVariantUtil*.

## Installation

Download the latest release of *StructuralVariantUtil* by running the following command in R. The installation should finish within a few minutes.

```
# Download StructuralVariantUtil
devtools::install_github("tgong1/StructuralVariantUtil")
# Load the package
library(StructuralVariantUtil)
```

## Third Party Software

Some functions of *StructuralVariantUtil* require *Shiny-SoSV* [1] and *bedtools* [2].

Download the latest release of *Shiny-SoSV* by cloning the repository:

```
git clone https://github.com/tgong1/Shiny-SoSV.git
```

Download *bedtools* and install *bedtools* (<https://bedtools.readthedocs.io/en/latest/content/installation.html>). Here shows one way to download from *bedtools* GitHub:

```
wget https://github.com/arq5x/bedtools2/releases/download/vx.xx.x/bedtools-
x.xx.x.tar.gz
tar -zxvf bedtools-x.xx.x.tar.gz
cd bedtools2
make
```

## Main Utilities

### SV calling performance estimation

*StructuralVariantUtil* estimates somatic SV calling performance including precision, sensitivity and F1 score, from whole genome sequencing data, based on prediction models developed in *ShinySoSV* [1]. Input parameters required include any candidate caller(s) (*Manta*, *Lumpy*, *GRIDSS*, *Delly*, *SvABA*) and variables impacting somatic SV calling, including variant

allele frequency (VAF), normal coverage, tumour coverage and tolerance of breakpoint precision.

We require the following input parameters from your study to be stored in a data.frame.

- sampleID (character): unique identifier of sample(s). The following example demonstrates the SV calling performance prediction for a cohort of 100 samples.
- VAF (numeric): variant allele frequency or estimated tumour purity
- N\_coverage (numeric): depth of coverage of normal sample
- T\_coverage (numeric): depth of coverage of tumour sample
- BND\_threshold (numeric): tolerance of breakpoint precision. The predicted sensitivity and precision will be higher if you have higher tolerance of breakpoint precision, i.e. higher *BND\_threshold*.

Other input parameters required include performance measurement(s) to estimate (*sensitivity, precision, F1\_score*) for any call set(s) (*individual, union, intersection*). We demonstrate SV calling performance estimation here using a simulated data set as input, stored in data.frame *newdata*. Then we predict F1 score of five SV callers' calling performance based on the input variables.

```
set.seed(1)
newdata <- data.frame(sampleID = paste0("sample_",c(1:100)), VAF =
round(rnorm(100, mean=0.5, sd=0.1),digits = 2), N_coverage = round(rnorm(100,
mean=30, sd=10),digits = 2), T_coverage = round(rnorm(100, mean=60,
sd=10),digits = 2), BND_threshold = 100)
head(newdata)
  sampleID  VAF N_coverage T_coverage BND_threshold
1 sample_1 0.44      23.80      64.09           100
2 sample_2 0.52      30.42      76.89           100
3 sample_3 0.42      20.89      75.87           100
4 sample_4 0.66      31.58      56.69           100
5 sample_5 0.53      23.45      37.15           100
6 sample_6 0.42      47.67      84.98           100
performance <- "F1_score"
callset <- "individual"
candidate_callers <- c("Manta","Lumpy","GRIDSS","Delly","SvABA")
```

Once all input data has been loaded, we proceed to run the function *ShinySoSV\_prediction*, which outputs data.frame with value of predicted performance (e.g. F1 score in this example) for all selected caller(s) and/or their pairwise union and intersection sets. Please note the *ShinySoSV* tool must be downloaded in the same directory when running *ShinySoSV\_prediction* in R.

```
df_prediction <- ShinySoSV_prediction(Candidate_callers, newdata, performance,
callset)
head(df_prediction)
  sampleID VAF N_coverage T_coverage BND_threshold fit_F1_score_Manta
fit_F1_score_Lumpy fit_F1_score_GRIDSS
1 sample_1 0.44      23.80      64.09           100      0.9019996
0.8416341      0.8575614
2 sample_2 0.52      30.42      76.89           100      0.8850921
0.8359865      0.8291313
3 sample_3 0.42      20.89      75.87           100      0.9056688
0.8452554      0.8640465
4 sample_4 0.66      31.58      56.69           100      0.8953312
0.8383520      0.8328662
5 sample_5 0.53      23.45      37.15           100      0.8712206
0.8140931      0.8192697
6 sample_6 0.42      47.67      84.98           100      0.9185124
0.8469072      0.8561267
  fit_F1_score_Delly fit_F1_score_SvABA
1      0.7229912      0.7674706
2      0.7320435      0.7206198
3      0.7244832      0.7801245
4      0.7614952      0.7334944
5      0.7138015      0.7284613
6      0.7279539      0.7773021
```

## Convert VCF format to a R data frame

The function *vcf\_to\_dataframe* can be used to convert SVs in VCF format to a R data frame. This function requires input as the file path to a VCF file.

The following example shows the use of function *vcf\_to\_dataframe* on VCF output from Manta (v1.4.0) ran on a pair of simulated tumour and normal BAM [1]. This function is expected to work for VCF format v4.1 or above, and has been tested on VCF output files from Manta, GRIDSS, Lumpy, Delly and SvABA.

This function outputs a data.frame with the following variables:

- CHROM: chromosome of the first breakpoint; CHROM field in VCF
- POS: genomic location of the first breakpoint; POS field in VCF
- ID\_caller: ID field in VCF
- REF: reference allele; REF field in VCF
- ALT: alternate allele; ALT field in VCF
- QUAL: quality score; QUAL field in VCF
- FILTER: FILTER field in VCF

- INFO\_END: END in INFO field in VCF
- INFO\_SVTYPE: SVTYPE in INFO field in VCF
- INFO\_SVLEN: SVLEN in INFO field
- INFO\_STRANDS: STRANDS in INFO field
- INFO\_CT: CT in INFO field
- INFO\_INV5: INV5 in INFO field
- INFO\_INV3: INV3 in INFO field
- INFO\_MATEID\_caller: MATEID in INFO field in VCF reported by caller

All other fields in the VCF file are ignored.

```
vcf_file <- system.file("extdata", "manta_sample1.vcf", package =
"StructuralVariantUtil")

df <- vcf_to_dataframe(vcf_file = vcf_file)
head(df)
```

CHROM	POS	ID_caller	REF
1	chr1 55008308	MantaINS:469:0:0:0:0	C
2	chr1 61988078	MantaINV:533:0:0:1:1:0	G
3	chr2 68529703	MantaINS:2496:0:0:0:0:0	C
4	chr2 119653368	MantaBND:2900:0:1:0:0:0:1	A
5	chr2 199398206	MantaINV:3544:0:0:1:3:0	A
6	chr2 212506915	MantaBND:1725:0:1:0:0:0:0	C

```
ALT QUAL FILTER
1
CATGGGGCAGGATGGCCATATTGGCCGGGGTGATGTGGAGGGCTTCCTAGAGGAACAGACATTGGAGCCGAGGCCTGAGG
TCAAGTTTATAACTTTCCTCT NA PASS
2
<INV> NA PASS
3
CTTTCTTATCAACTCCAACTTACAGGGTGAAGTTAGCCATCTCTTTCAGT NA PASS
4
A[CHR3:58161076[ NA PASS
5
<INV> NA PASS
6
C[CHR1:224467224[ NA PASS
INFO_END INFO_SVTYPE INFO_SVLEN INFO_STRANDS INFO_CT INFO_INV5 INFO_INV3
INFO_MATEID_caller
1 55008308 INS 100 NA NA FALSE FALSE
<NA>
2 61988179 INV 101 NA NA TRUE FALSE
<NA>
3 68529703 INS 50 NA NA FALSE FALSE
<NA>
4 NA BND NA NA NA FALSE FALSE
MantaBND:2900:0:1:0:0:0:0
5 199398307 INV 101 NA NA TRUE FALSE
<NA>
6 NA BND NA NA NA FALSE FALSE
MantaBND:1725:0:1:0:0:0:1
```

## Simple SV type classification

The function *simple\_SVTYPE\_classification* converts SVs in VCF format into pairs of BNDs at each fusion junction, and classifies them into one of five simple SV types, including deletion (DEL), duplication (DUP), insertion (INS), inversion (INV) and inter-chromosomal translocation (TRA).

The only required input parameter to this function is either the file path to a VCF file or SV VCF loaded as a R data frame. For the latter, the data frame must have at least the following variables (see section *Convert VCF format to data.frame in R*):

- CHROM: chromosome of the first breakpoint; CHROM field in VCF
- POS: genomic location of the first breakpoint; POS field in VCF
- ALT: alternate allele; ALT field in VCF reported by caller following the format described in VCF specification (Version 4.1 above) section 5.4

This function is expected to work for VCF format v4.1 or above, and has been tested on VCF output files from Manta, GRIDSS, Lumpy, Delly and SvABA.

Here, we demonstrated the use of this function using a VCF output file from Manta (v1.4.0).

```
vcf_file <- system.file("extdata", "manta_sample1.vcf", package =
"StructuralVariantUtil")
bedpe <- simple_SVTYPE_classification(SV_data = vcf_file, caller_name="manta")
head(bedpe)
```

	chrom1	start1	end1	chrom2	start2	end2	SVTYPE	strand1	strand2
ID	ID_mate								
1	chr1	55008307	55008308	chr1	55008307	55008308	INS	<NA>	<NA>
manta_1_1_1	manta_1_2_1								
2	chr1	61988077	61988078	chr1	61988178	61988179	INV	-	-
manta_2_1_2	manta_2_2_2								
3	chr2	68529702	68529703	chr2	68529702	68529703	INS	<NA>	<NA>
manta_3_1_3	manta_3_2_3								
4	chr2	119653367	119653368	chr3	58161075	58161076	TRA	+	-
manta_4_1_4	manta_4_2_4								
5	chr2	199398205	199398206	chr2	199398306	199398307	INV	-	-
manta_5_1_5	manta_5_2_5								
6	chr2	212506914	212506915	chr1	224467223	224467224	TRA	+	-
manta_6_1_6	manta_6_2_6								

ALT	ID_caller	REF	QUAL
1			
CATGGGGCAGGATGGCCATATTGGCCGGGGTGATGTGGAGGGCTTCCTAGAGGAACAGACATTGGAGCCGAGGCCTGAGG			
TCAAGTTTATAACTTTCCTCT	Manta	INS:469:0:0:0:0:0	C NA

```

2
<INV>    MantaINV:533:0:0:1:1:0    G    NA
3
CTTTCTTATCAACTCCAACTTACAGGGTGAAGTTAGCCATCTCTTTCAGT    MantaINS:2496:0:0:0:0:0
C    NA
4
A[CHR3:58161076[    MantaBND:2900:0:1:0:0:0:1    A    NA
5
<INV>    MantaINV:3544:0:0:1:3:0    A    NA
6
C[CHR1:224467224[    MantaBND:1725:0:1:0:0:0:0    C    NA
    FILTER INFO_SVTYPE INFO_SVLEN INFO_STRANDS INFO_CT INFO_INV5 INFO_INV3
INFO_MATEID_caller
1    PASS            INS            100            NA            NA            FALSE            FALSE
<NA>
2    PASS            INV            101            NA            NA            TRUE             FALSE
<NA>
3    PASS            INS            50             NA            NA            FALSE            FALSE
<NA>
4    PASS            BND            NA             NA            NA            FALSE            FALSE
MantaBND:2900:0:1:0:0:0:0
5    PASS            INV            101            NA            NA            TRUE             FALSE
<NA>
6    PASS            BND            NA             NA            NA            FALSE            FALSE
MantaBND:1725:0:1:0:0:0:1

```

The output SV calls are in BEDPE format with variables as described in **Table 1** for each SV event. Additional variables (except CHROM and POS) in the input data frame are appended to the output unchanged. SVTYPE in the output is identical to SVTYPE in INFO field of the input VCF if the values were already DEL, DUP, INS, INV and TRA. If input SVTYPE is BND, simple SV types are derived from ALT filed in VCF (**Table 2**). If the ALT field of the input VCF does not follow BND format but contain an inserted sequence or has the <INS> designation, INS is assigned as the simple SV type.

Table 1. The BEDPE format of SV fusion junctions

Variables	VCF fields <sup>1</sup>	Note
chrom1	#CHROM	The name of the chromosome of first breakend
start1	POS	The zero-based starting position of first breakend on chrom1
end1	POS	The one-based ending position of first breakend on chrom1
chrom2	#CHROM or ALT	chromosome of second breakend

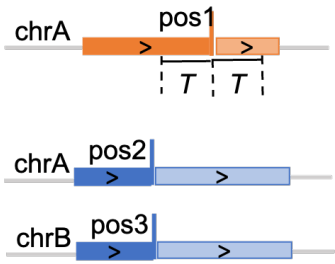
start2	ALT or INFO: END	The zero-based starting position of second breakend on chrom2
end2	ALT or INFO: END	The one-based starting position of second breakend on chrom2
Strand1	INFO: STRANDS (e.g. Lumpy); CT=3to3 or 5to5 (e.g. Delly); INV5 or INV3 (e.g. Manta) or ALT <sup>2</sup>	strand of first breakend
Strand2		strand of second breakend
SVTYPE	INFO: SVTYPE or ALT <sup>2</sup>	Simple SV types, based on strands
ID <sup>3</sup>	NA	Unique identifier of first breakend
ID_mate <sup>3</sup>	NA	Unique identifier of second breakend

<sup>1</sup>Information source from input VCF.

<sup>2</sup>see Table 2.

<sup>3</sup>Unique identifiers assigned by *StructuralVariantUtil*.

Table 2. Strands and SV type derivation in reference to ALT field in VCF

Reference	sample	ALT field in VCF	Simple SV type identified	Strands identified
	chrA:pos1 chrA:pos2	t[p[	DEL	+-
	chrA:pos2 chrA:pos1	]p]t	INS	-+
	chrA:pos1 chrA:pos2	t]p]	INV	++
	chrA:pos2 chrA:pos1	[p[t		--
	chrA:pos1 chrB:pos3	Any ALT	Inter-chromosomal TRA	Any strands
	chrA:pos1 chrA:pos1		INS	

The output data frame of *simple\_SVTYPE\_classification* contains all required information for presenting the SVs in a CIRCOS plot using the *circlize* R package (demonstrated below, **Figure 1**). The function *prepare\_SV\_for\_circos* can be used to prepare the output from *simple\_SVTYPE\_classification* for generating CIRCOS plots with R/circlize.



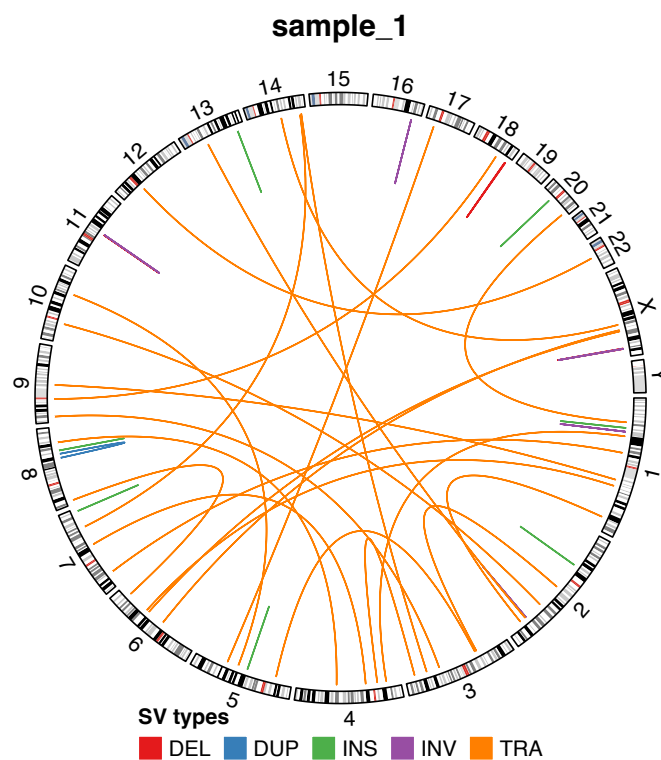
Here we demonstrate the CIRCOS plot drawing for this example SV call set (**Figure 1**).

```
SVdata_for_circos <- prepare_SV_for_circos(bedpe)

SVColours <- c(RColorBrewer::brewer.pal(n = 5, name = 'Set1'))
names(SVColours) <- c("DEL", "DUP", "INS", "INV", "TRA")

circlize::circos.initializeWithIdeogram(species = "hg38", plotType =
c("ideogram", "labels"))
circlize::circos.genomicLink(SVdata_for_circos[[1]], SVdata_for_circos[[2]],
                             col =
SVColours[match(SVdata_for_circos[[1]]$SVTYPE, names(SVColours))],h=0.2)
circlize::circos.genomicLink(SVdata_for_circos[[3]],SVdata_for_circos[[4]],
                             col =
SVColours[match(SVdata_for_circos[[3]]$SVTYPE, names(SVColours))])
title(main = "sample_1")

lgd_links = ComplexHeatmap::Legend(at = c("DEL", "DUP", "INS", "INV", "TRA"),
legend_gp = grid::gpar(fill=SVColours), type = "grid",by_row = TRUE,ncol=5,
                             title_position = "topleft", title = "SV")
ComplexHeatmap::draw(lgd_links,
                     y = ggplot2::unit(ComplexHeatmap::height(lgd_links),
"mm"))
circlize::circos.clear()
```



**Figure 1.** CIRCOS plot of the example SV call set

## SV caller Integration filtration

Function *SV\_integration* allows users to integrate SV call sets, provided in VCF format (e.g. results from different SV callers). The function does this by first standardising BND pairs in simple SV types then merging SVs based on BND positions. The function outputs integrated SV calls as a R data frame in bedpe format.

Input parameters required to run *SV\_integration* include:

- `vcf_files`: list of VCF files in the same order as `SVCaller_name`
- `SVCaller_names`: vector of names of SV callers or unique identifier of the VCFs
- `sampleID`: unique identifier for the sample. Default as “sample\_1”.
- `bkpt_T_callers`: threshold of breakpoint position difference between two calls to be concordant. Default is 100bp.
- `PASS_filter`: filtering based on FILTER field of two calls in VCF: “both” to require both two calls with “PASS”, “one” to require one of the two calls with “PASS”, “none” to ignore this filtering. Default is “both”.
- `SVTYPE_ignore`: whether to consider same SV type for SV concordance. TRUE or FALSE.
- `bedtools_dir`: path of bedtools. Specify your path here or add its path to system PATH.

```
# This example SV integration should finish in a few minutes.
# two example VCFs from two different SV callers (Manta & GRIDSS)
vcf_files<- c(system.file("extdata",
"manta_SVEngine_TumorSV2.60x_NormalSV1.60x_0.5.T.PASS.recode.vcf", package =
"StructuralVariantUtil"),
              system.file("extdata",
"GRIDSS_SVEngine_TumorSV2.60x_NormalSV1.60x_0.5_somatic_PASS_annotated.vcf",
package = "StructuralVariantUtil"))

# optional sample name
sampleID <- "sample_1"

# mandatory character vector of SV call names
SVCaller_names <- c("manta","gridss")

# SV integration
integrated_bedpe <- SV_integration(vcf_files, SVCaller_names, sampleID)
head(integrated_bedpe)
  chrom1 start1 end1 chrom2 start2 end2 SVTYPE strand1 strand2
ID ID_mate
```

```

1 chr1 1615683 1615684 chr1 1615683 1615684 INS <NA> <NA>
all_1_1_1 all_1_2_1
2 chr1 1615683 1615684 chr19 16196780 16196781 TRA + -
all_2_1_2 all_2_2_2
4 chr1 6170587 6170588 chr22 23815023 23815024 TRA - +
all_4_1_4 all_4_2_4
5 chr1 6170587 6170588 chr1 6170687 6170688 DEL + -
all_5_1_5 all_5_2_5
7 chr1 7572789 7572790 chr1 8572789 8572790 DEL + -
all_7_1_7 all_7_2_7
8 chr1 8065641 8065642 chr16 62560615 62560616 TRA + -
all_8_1_8 all_8_2_8

```

```

ALT ID_caller
1
TAAATAGCTAGGTGTGGTGGCACATGCCTGTAATCCCAGCCACTTGAGAGGCTGACACACAAGAGAATCACTTGAACC
CAGGAGGCAGAGGTTGCAGTG MantaINS:4:0:0:0:0
2
T[CHR19:16196781[ MantaBND:4:0:1:0:0:0:1
4
]CHR22:23815024]T MantaBND:30:0:1:0:0:0:1
5
T MantaDEL:30:0:0:0:1:0
7
<DEL> MantaDEL:41:0:1:0:0:0
8
A[CHR16:62560616[ MantaBND:45:0:1:0:0:0:0

```

```

REF QUAL FILTER INFO_SVTYPE INFO_SVLEN INFO_STRANDS
1
T NA PASS INS 100 NA
2
T NA PASS BND NA NA
4
T NA PASS BND NA NA
5
TGCTTGGGGCTCCCACACAGGGAGGGCACCCCTGTGGAGGGCTAGGGCACACAGGGGAGCCAGCAGCAAGGGCCGCCCCAG
GTGGGTTTATGTGGGTGAGGC NA PASS DEL -100 NA
7
A NA PASS DEL -1000000 NA
8
A NA PASS BND NA NA
INFO_CT INFO_INV5 INFO_INV3 INFO_MATEID_caller manta_ID
gridss_ID
1 NA FALSE FALSE <NA> manta_1_1_1
<NA>
2 NA FALSE FALSE MantaBND:4:0:1:0:0:0:0 manta_2_1_2,manta_3_1_3
gridss_1_1_1,gridss_2_1_2
4 NA FALSE FALSE MantaBND:30:0:1:0:0:0:0 manta_4_1_4,manta_6_1_6
gridss_4_1_4,gridss_5_1_5
5 NA FALSE FALSE <NA> manta_5_1_5
gridss_3_1_3
7 NA FALSE FALSE <NA> manta_7_1_7
gridss_6_1_6
8 NA FALSE FALSE MantaBND:45:0:1:0:0:0:1 manta_8_1_8
gridss_7_1_7

```

Integrated SV output from *SV\_integration* can be used as input to the R/ VennDiagram package to generate a Venn diagram of SV calls from the input SV callsets, as shown in **Figure 2**.

```
union_ID <- integrated_bedpe[, (ncol(integrated_bedpe)-
length(SVCaller_names)+1):ncol(integrated_bedpe)]
for(i in c(1: length(SVCaller_names))){
  index <- rowSums(!is.na(union_ID)) != 1 & (!is.na(union_ID[,i]))
  union_ID[index,i] <- integrated_bedpe[index,]$ID
  assign(SVCaller_names[i], union_ID[,i][!is.na(union_ID[,i])])
}
x <- do.call("list", lapply(SVCaller_name,function(s) eval(parse(text=s))))
g <- VennDiagram::venn.diagram(
  x = x,
  category.names = SVCaller_name,
  # Circles
  lwd = 2,
  fill = RColorBrewer::brewer.pal(3, "Pastel2")[1:length(SVCaller_name)],

  # Numbers
  cex = 1,
  fontface = "bold",
  filename = NULL,
  output=FALSE
)
grid::grid.newpage()
grid::grid.draw(g)
```

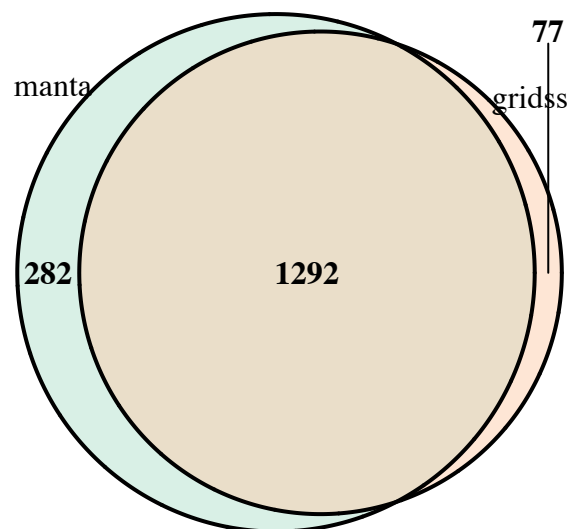


Figure 2. Venn diagram of the integrated SV call set

## Spectrum of SV types for large cohort

The function *spectrum\_SV\_type* can be used to summarize the spectrum of SV types at a cohort level using function *spectrum\_SV\_type*. For a large cancer cohort (recommended  $n > 30$ ), the function can also identify hyper-SV mutated tumour samples.

Input parameters required for *spectrum\_SV\_type* include a character vector of sample ID and a list of SV data, which can be a list of VCF file paths or a list of data frames of SV callsets, for all samples in a cohort. To use a list of SV data frames as input, see sections of ***Convert VCF format to data.frame in R*** and ***Simple SV type classification***. Function *spectrum\_SV\_type* outputs a data.frame summarising the number of each SV type for each input sample.

```
# create character vector of three sample identifiers
All_sampleID <- paste0("sample",c(1,2,3))
# identify three example VCFs
vcf_files <- c(system.file("extdata", "manta_sample1.vcf", package =
"StructuralVariantUtil"),
               system.file("extdata", "manta_sample2.vcf", package =
"StructuralVariantUtil"),
               system.file("extdata", "manta_sample3.vcf", package =
"StructuralVariantUtil"))

# summarise SV types
Spectrum_SVTYPE <- spectrum_SV_type(All_sampleID, All_SV_data = vcf_files)
Spectrum_SVTYPE
  sampleID INS INV TRA DUP DEL
1 sample1   7   5  24   1   3
2 sample2   2   2  30   5  11
3 sample3  11  17  53   5  13
```

To estimate hyper-SV analysis, set the argument *identify\_hyperSV\_tumour* to TRUE. Additionally, provide the optional arguments:

- *identify\_hyperSV\_tumour*: TRUE or FALSE. Whether to identify hyper-SV mutated tumour samples for large cancer cohort. Default as FALSE.
- *threshold\_total*: threshold of minimum total count of SVs per sample. The default value is the average total SV count in the cohort.
- *threshold\_relative\_freq*: the threshold of minimum relative frequency of one SV type per sample. The default value is 50%.

Setting *identify\_hyperSV\_tumour* to TRUE will cause *spectrum\_SV\_type* to output a list of two data frames. The first will be a data frame of counts of SV types (as for *identify\_hyperSV\_tumour* = FALSE) and a second data frame with variables:

- *sampleID*: ID of sample with hyper-SV identified.
- *count*: count of SVs in this type
- *relative\_freq*: relative frequency of the particular SV type per sample.

- `total_count`: total count of SVs in this sample.
- `HYPER_SVTYPE`: which type of hyper-SV mutation.

In the example below, we load the test input SV callsets, in a data frame, for 100 samples, with breakpoints simulated randomly based on Hg38 genomic positions, excluding gap, centromere and telomere regions.

```
data(list)
All_sampleID <- paste0("sample_",c(1:100))

results <- spectrum_SV_type(All_sampleID, All_SV_data =list,
identify_hyperSV_tumour = TRUE)

# First output list object: counts of each SV type
Spectrum_SVTYPE <- results[[1]]
head(Spectrum_SVTYPE)
sampleID DEL DUP INS INV TRA
1 sample_1 167 300 33 164 248
2 sample_2 129 49 4 99 198
3 sample_3 299 242 53 163 204
4 sample_4 270 246 86 148 167
5 sample_5 187 247 89 150 289
6 sample_6 85 239 24 121 244

# Second output list object: hyper-SV results
Hyper_SV_sample <- results[[2]]
Hyper_SV_sample
  sampleID count relative_freq total_count HYPER_SVTYPE
96 sample_96  982      0.6323245      1553    hyper-DEL
97 sample_97  826      0.5772187      1431    hyper-DEL
98 sample_98  609      0.6017787      1012    hyper-DEL
99 sample_99  854      0.5685752      1502    hyper-DEL
100 sample_100 806      0.6494762      1241    hyper-DEL
```

## Spectrum of SV breakpoints in genomic bins

In a cohort analysis, genomic regions recurrently impacted by SVs or specific SV types may be of interest, such genome-wide pattern of SV hotspots is sometimes known as the spectrum of SVs. The function *spectrum\_SV\_breakpoint* can be used for identifying and visualising the spectrum of SVs, by summing the number of SV breakpoints within 1 Mb non-overlapping bin across the genome.

Input parameters required by *spectrum\_SV\_breakpoint* include a character vector of sample ID and SV data, which can be a list of VCF file paths or a list of data frames of SV callsets, one

for each sample in the cohort. The list of callsets can be provided as a list of VCF file paths or a list of SV data frames for all samples in the cohort, similar to the input required for *summary\_SV\_type*. SV hotspots are defined as genomic regions most frequently ( $> Q_3 + k \times (Q_3 - Q_1)$ ) affected by SV breakpoints, either in the same genome or recurrent across genomes.

The required input thresholds to define SV hotspots are:

- *threshold\_count\_breakpoint*: the  $k$  value in Tukey's fences approach to find outliers in breakpoint count. The default is set as 1.5. Considering clustered SV breakpoints such as chromothripsis can be attained in a single tumour, it can be more stringent on this threshold by setting this threshold to be higher (for example, as 3).
- *threshold\_count\_sample*: the  $k$  value in Tukey's fences approach to define outliers for sample count. The default is set as 1.5.

Function *spectrum\_SV\_breakpoint* outputs a data.frame with the following variables.

- *chrom\_bin\_labels* (character): unique identifier of genomic bin
- *bin\_labels* (character): ID of bin for each chromosome
- *bin* (character): genomic region of bin
- *breaks* (numeric): start of genomic bin
- *sampleID* (character): sample name or ID
- *chrom* (character): chromosome of the breakpoint in the genomic bin
- *pos* (numeric): genomic location of the breakpoint in the genomic bin
- *count\_breakpoints* (numeric): the count of SV breakpoint in the genomic bin
- *count\_sample* (numeric): the number of samples with at least one SV breakpoint in the genomic bin
- *is\_hotspot\_breakpoint* (TRUE or FALSE): whether defined as hotspot based on count of breakpoints ( $> Q_3 + \text{threshold\_count\_breakpoint} \times (Q_3 - Q_1)$ )
- *is\_hotspot\_sample* (TRUE or FALSE): whether defined as hotspot based on count of samples ( $> Q_3 + \text{threshold\_count\_sample} \times (Q_3 - Q_1)$ )
- *is\_hotspot* (TRUE or FALSE): whether defined as hotspot either in the same genome or recurrent across genomes

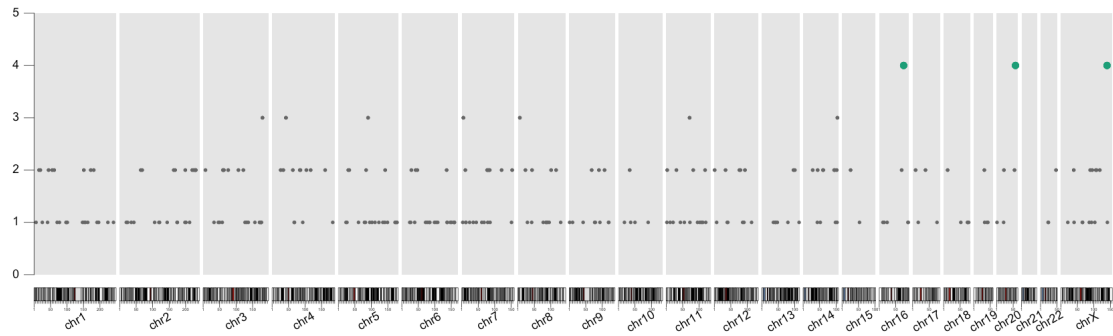
```
All_sampleID <- paste0("sample",c(1,2,3))
```

```
vcf_files <- c(system.file("extdata", "manta_sample1.vcf", package =
"StructuralVariantUtil"),
              system.file("extdata", "manta_sample2.vcf", package =
"StructuralVariantUtil"),
              system.file("extdata", "manta_sample3.vcf", package =
"StructuralVariantUtil"))

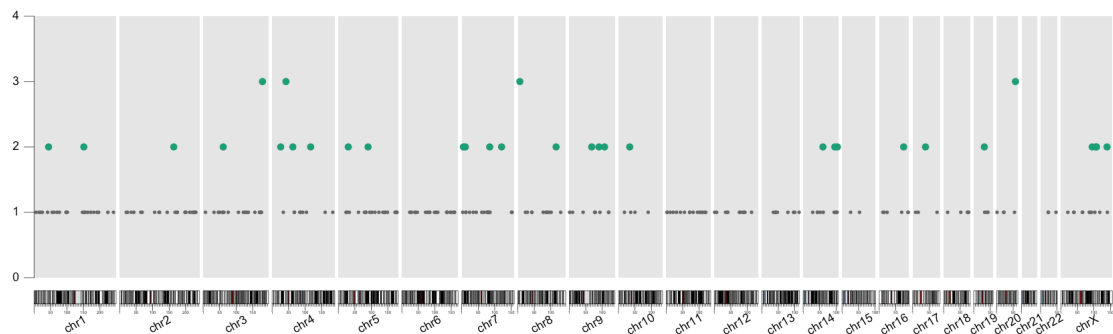
df_bin_all_hotspots <- spectrum_SV_breakpoint(All_sampleID, All_SV_data =
vcf_files)
head(df_bin_all_hotspots)
  chrom_bin_labels bin_labels          bin  breaks sampleID chrom
pos                               ID
1      chr1_56         56  (5.5e+07,5.6e+07] 5.50e+07 sample1 chr1
55008308 StructuralVariantUtil_1_1_1
2      chr1_62         62  (6.1e+07,6.2e+07] 6.10e+07 sample1 chr1
61988078 StructuralVariantUtil_2_1_2
41     chr1_56         56  (5.5e+07,5.6e+07] 5.50e+07 sample1 chr1
55008308 StructuralVariantUtil_1_2_1
42     chr1_62         62  (6.1e+07,6.2e+07] 6.10e+07 sample1 chr1
61988179 StructuralVariantUtil_2_2_2
46     chr1_225        225 (2.24e+08,2.25e+08] 2.24e+08 sample1 chr1
224467224 StructuralVariantUtil_6_2_6
49     chr1_71         71  (7e+07,7.1e+07] 7.00e+07 sample1 chr1
70505412 StructuralVariantUtil_9_2_9
  count_breakpoints count_sample is_hotspot_breakpoint is_hotspot_sample
is_hotspot
1                2            1             FALSE             FALSE
FALSE
2                2            1             FALSE             FALSE
FALSE
41               2            1             FALSE             FALSE
FALSE
42               2            1             FALSE             FALSE
FALSE
46               1            1             FALSE             FALSE
FALSE
49               1            1             FALSE             FALSE
FALSE
```

Additionally the function *spectrum\_SV\_breakpoint* produces two figures, one showing the number of breakpoints at each genomic sites and another showing the number of samples in the cohort harbouring a breakpoin at each genomic site (**Figure 3** and **4**). In both figures, SV hotspots are highlighted in green.





**Figure 3. SV frequency genome-wide for all samples, showing the number of breakpoints. Hotspot regions are highlighted as green.**



**Figure 4. SV frequency genome-wide for all samples, showing the number of samples. Hotspot regions are highlighted as green.**

## Copy number changes of deletion and duplication events

DEL and DUP SV types can result in copy number loss and gains respectively. However, the actual number of DNA fragment copies in a DEL or DUP event is not available from SV calling results. CNV callers are developed with a focus on inferring and reporting copy number changes, while SV callers infer SV breakpoint and type and rarely attempt to infer copy number changes resulting by SV events. To better understand the DELs and DUPs, *StructuralVariantUtil* facilitates integrating DEL and DUP with copy number segments and categorise DEL and DUP based on copy number values with function *SV\_CNV\_integration*.

The input parameters required by function *SV\_CNV\_integration* include SV and CNV data in data.frame. Prepare your SV data.frame using function *simple\_SVTYPE\_classification* or prepare the input in bedpe format with at least the following variables:

- `chrom1(character)`: chromosome of the first breakpoint
- `start1(numeric)`: The zero-based starting position of first breakend on `chrom1`
- `end1(numeric)`: The one-based starting position of first breakend on `chrom1`
- `chrom2(character)`: chromosome of the second breakpoint
- `start2(numeric)`: The zero-based starting position of second breakend on `chrom2`
- `end2(numeric)`: The one-based starting position of second breakend on `chrom2`
- `ID(character)`: ID of SV first breakpoint
- `ID_mate(character)`: ID of SV second breakpoint
- `SVTYPE(character)`: SV types

```
vcf_file = system.file("extdata", "manta_sample4.vcf", package =
"StructuralVariantUtil")
SV_data <- simple_SVTYPE_classification(vcf_file, caller_name = "manta")
head(SV_data)
```

Load the test input CNV data as following. The input CNV data is required to be stored in `data.frame` with at least the following columns:

- `chrom(character)`: chromocome
- `start (numeric)`: start position for the CN segment
- `end(numeric)` end position for the CN segment
- `cn (numeric)`: integer total copy number (e.g. 2 for copy number neutral/unaltered region)

```
data(CNV_bed)
head(CNV_bed)
  chromosome  start    end  cn
1      chr1  10000  905503   2
2      chr1  905503 1374792   2
3      chr1 1374792 1943930   2
4      chr1 1943930 2423204   2
5      chr1 2423204 2796280   2
6      chr1 2796280 3446159   2
```

Other input parameters include *sample\_ID* (sample name or ID) and *overlap\_f* (the fraction of minimum overlap required of CNV segment as a fraction of SV, default as 0.5). In addition, define parameter of *bedtools\_dir* to provide path of bedtools by specifying your path or adding its path to system PATH.

Once the input data and parameters have been loaded, we proceed to run the function *SV\_CNV\_integration*, which results a data.frame with deletions and duplications and their overlapping discrete copy number segments.

- sampleID: Unique identifier of sample
- SV\_chrom: chromosome of SV
- SV\_start: start position of DEL or DUP region
- SV\_end: end position of DEL or DUP region
- SV\_ID: ID of SV
- SV\_type: type of SV
- CNV\_chrom: CNV chromosome
- CNV\_start: CNV start position
- CNV\_end: CNV end position
- CNV\_cn: copy number value
- overlap: the number of bases overlapping between SV region and CNV region.

```
sampleID <- "sample_4"
SV_CNV_integrated <- SV_CNV_integration(sampleID, SV_data, CNV_data = CNV_bed,
bedtools_dir = "/opt/homebrew/bin/bedtools")
head(SV_CNV_integrated)
```

	sampleID	SV_chrom	SV_start	SV_end	SV_ID	SV_type	CNV_chrom
			CNV_start	CNV_end	CNV_cn	overlap	
1	sample_4	chr2	190055341	190534896	Manta_105_1_93	DUP	chr2
			190052119	190552133		2	479555
2	sample_4	chr3	83218893	90328953	Manta_135_1_117	DEL	chr3
			83222254	90772459		1	7106699
3	sample_4	chr3	140607229	140793386	Manta_172_1_153	DUP	chr3
			140605567	140795567		3	186157
4	sample_4	chr3	161840785	167251478	Manta_173_1_154	DEL	chr3
			162905564	166285563		2	3379999
5	sample_4	chr4	29153967	29368058	Manta_187_1_166	DEL	chr4
			29149724	29369689		1	214091
6	sample_4	chr4	61828504	61869283	Manta_201_1_179	DEL	chr4
			61831419	61871420		1	37864

## SV breakpoint gene annotation

StructuralVariantUtil facilitates SV breakpoints annotation based on gene regions using function *SV\_gene\_annotation*. In addition, it summarized all gene fusion pairs resulted from SVs and provides a table of gene fusion frequency in the sample or cohort. A SV breakpoint is annotated as interrupting a gene if it is within the gene region.

Firstly, provide the input vcf file or use *simple\_SVTYPE\_classification* to format VCF to SV BND pairs in bedpe format with at least following variables:

- **chrom1(character)**: chromosome of the first breakpoint
- **start1(numeric)**: The zero-based starting position of first breakend on chrom1
- **end1(numeric)**: The one-based starting position of first breakend on chrom1
- **chrom2(character)**: chromosome of the second breakpoint
- **start2(numeric)**: The zero-based starting position of second breakend on chrom2
- **end2(numeric)**: The one-based starting position of second breakend on chrom2
- **ID(character)**: ID of SV first breakpoint
- **ID\_mate(character)**: ID of SV second breakpoint

```
vcf_file <- system.file("extdata", "manta_sample1.vcf", package =  
"StructuralVariantUtil")
```

The provided bed file of gene regions (ensembl\_release99\_gene.bed) was derived based on ENSEMBL gene annotation release version 99 for GCRh38. User can use their preferred gene database and prepare it in data.frame with at least following variables:

- **chrom**: Reference sequence chromosome of the gene
- **start**: gene or transcription start position
- **end**: gene or transcription end position
- **gene\_name**: name or unique identifier of gene

```
gene_file = system.file("extdata", "ensembl_release99_gene.bed", package =  
"StructuralVariantUtil")  
gene_bed <- read.table(gene_file, header=TRUE)
```

Make sure you have also defined parameter of *bedtools\_dir* to provide path of bedtools by specifying your path or adding its path to system PATH. Once the input data have been loaded, we proceed to run the function *SV\_gene\_annotation*, which outputs a data.frame with interrupted genes added to the input SV data.frame:

- **pos1\_overlap\_gene**: name of gene interrupted by first breakpoint (chr1 and pos1 in input data.frame)
- **pos2\_overlap\_gene**: name of gene interrupted by second position (chr2 and pos2 in input data.frame)

```
results <- SV_gene_annotation(SV_data = vcf_file, gene_bed)  
SV_geneAnnotated <- results[[1]]
```

	chrom1	start1	end1	chrom2	start2	end2	SVTYPE	strand1	strand2
1	chr1	55008307	55008308	chr1	55008307	55008308	INS	<NA>	<NA>
2	chr1	61988077	61988078	chr1	61988178	61988179	INV	-	-
3	chr2	68529702	68529703	chr2	68529702	68529703	INS	<NA>	<NA>
4	chr2	119653367	119653368	chr3	58161075	58161076	TRA	+	-
5	chr2	199398205	199398206	chr2	199398306	199398307	INV	-	-
6	chr2	212506914	212506915	chr1	224467223	224467224	TRA	+	-

	ID	ID_mate
1	manta_1_1_1	manta_1_2_1
2	manta_2_1_2	manta_2_2_2
3	manta_3_1_3	manta_3_2_3
4	manta_4_1_4	manta_4_2_4
5	manta_5_1_5	manta_5_2_5
6	manta_6_1_6	manta_6_2_6

ALT

1  
CATGGGGCAGGATGGCCATATTGGCCGGGGTGATGTGGAGGGCTTCCTAGAGGAACAGACATTGGAGCCGAGGCCTGAGG  
TCAAGTTTATAACTTTCTCT

2  
<INV>

3  
CTTTCTTATCAACTCCAACTTACAGGGTGAAGTTAGCCATCTCTTTCAGT

4  
A[CHR3:58161076[

5  
<INV>

6  
C[CHR1:224467224[

	ID_caller	REF	QUAL	FILTER	INFO_SVTYPE	INFO_SVLEN	INFO_STRANDS
1	MantaINS:469:0:0:0:0:0	C	NA	PASS	INS	100	NA
2	MantaINV:533:0:0:1:1:0	G	NA	PASS	INV	101	NA
3	MantaINS:2496:0:0:0:0:0	C	NA	PASS	INS	50	NA
4	MantaBND:2900:0:1:0:0:0:1	A	NA	PASS	BND	NA	NA
5	MantaINV:3544:0:0:1:3:0	A	NA	PASS	INV	101	NA
6	MantaBND:1725:0:1:0:0:0:0	C	NA	PASS	BND	NA	NA

	INFO_CT	INFO_INV5	INFO_INV3	INFO_MATEID_caller	pos1_overlap_gene
1	NA	FALSE	FALSE	<NA>	BSND
2	NA	TRUE	FALSE	<NA>	PATJ
3	NA	FALSE	FALSE	<NA>	APLF
4	NA	FALSE	FALSE	MantaBND:2900:0:1:0:0:0:0	CFAP221
5	NA	TRUE	FALSE	<NA>	SATB2
6	NA	FALSE	FALSE	MantaBND:1725:0:1:0:0:0:1	ERBB4

	pos2_overlap_gene
1	BSND
2	PATJ
3	APLF
4	FLNB
5	SATB2
6	CNIH3

In addition, function *SV\_gene\_annotation* output a table of summarised gene fusions resulted from SV, with the following variables:

- **pos1\_overlap\_gene**: name of gene interrupted by pos1
- **pos2\_overlap\_gene**: name of gene interrupted by pos2

- **gene\_fusions**: name of gene fusions
- **breakpoint\_count**: count of fusion events due to SVs
- **sample\_count**: count of samples having this gene fusion

```
df_summary_gene_fusions <- results[[2]]
head(df_summary_gene_fusions)
```

	pos1_overlap_gene	pos2_overlap_gene	gene_fusions	breakpoint_count
1	ABCD2	SPECC1L	ABCD2;SPECC1L	1
2	ABCD2	SPECC1L-ADORA2A	ABCD2;SPECC1L-ADORA2A	1
3	CFAP221	FLNB	CFAP221;FLNB	1
4	COL4A6	HACE1	COL4A6;HACE1	1
5	ERBB4	CNIH3	ERBB4;CNIH3	1
6	ESR1	XKR5	ESR1;XKR5	1

	sample_count
1	1
2	1
3	1
4	1
5	1
6	1

## REFERENCES

1. Gong T, Hayes VM, Chan EKF: **Shiny-SoSV: A web-based performance calculator for somatic structural variant detection.** *PLOS ONE* 2020, **15**:e0238108.
2. Quinlan AR, Hall IM: **BEDTools: a flexible suite of utilities for comparing genomic features.** *Bioinformatics* 2010, **26**:841-842.