RSVSim
an R/Bioconductor package for the simulation of structural variations

Christoph Bartenhagen

April 27, 2020

Contents

1 Introduction

The simulation of structural variations (SV) is an important measure to assess the performance of algorithms dealing with SVs and their detection and can help with the design of sequencing experiments. A simulation generates a base exact ground truth, which can be used to test the sensitivity and precision of SV callers.
A FASTA-file with the simulated, rearranged genome can be used by common, published read simulators (like [Huang *et al.*, 2011]), [Hu *et al.*, 2012]) to generate NGS datasets from various platforms that can then be used to asses an SV algorithm . A typical workflow consists of

SV simulation ⇒ (Paired-End) Read simulation ⇒ SV algorithm ⇒ Evaluation

Varying parameters of the SV simulation like SV type, size or location and of the read simulator like number of reads (coverage), insert-size (for paired-end) or read length can give helpful information for future sequencing experiment designs.

This package addresses the very first step of SV simulation and provides the following features:

- Simulation of deletions, insertions, inversions, tandem duplications and translocations (balanced and unbalanced) of various sizes
- Rearrangement of the human genome (hg19) by default or any other kind of genome available as FASTA file or *BSgenome* package
- Non-overlapping positioning of SV breakpoints within the whole genome or only a subset (e.g. coding, non-coding or low-complexity regions)
- Implementation of, e.g. previously detected or known, SVs at user-supplied coordinates
- Uniform distribution of SV breakpoints or simulation of biases towards repeat regions and regions of high homology according to different SV formation mechanisms (for hg19 only)
- Simulation of smaller mutations (SNPs and indels) close to the SV breakpoint
- Estimation of SV size distribution from real datasets
- Comparison of SV simulation with results from SV detection algorithms

1.1 Loading the package

After installation, the package can be loaded into R by typing

```
> library(RSVSim)
```

into the R console.
RSVSim requires the R-packages *Biostrings*, *IRanges*, *GenomicRanges* and *ShortRead*. Mainly for efficient and convenient storing and access of sequences and genomic coordinates. The packages *BSgenome.Hsapiens.UCSC.hg19*, *GenomicFeatures*, *rtracklayer* and *MASS* are suggested for certain functionalities.

2 Structural variation simulation

The main function for simulation is called *simulateSV*. The simulation works pretty similar for every different SV type by specifying number and size of the variation(s) and (optionally) the regions, where to place the variation (randomly or not). The size can be either one value for every SV type or a vector of values for every single SV.
The following sections give a short example for every SV type using a simple toy example with two chromosomes of 40bp each:

```
> genome = DNAStringSet(
 c("AAAAAAAAAAAAAAAAAAAATTTTTTTTTTTTTTTTTT",
 "GGGGGGGGGGGGGGGGGGGGCCCCCCCCCCCCCCCCCCCC"))
> names(genome) = c("chr1","chr2")
> genome
```

```
DNAStringSet object of length 2:
    width seq                                                                 names
[1]    40 AAAAAAAAAAAAAAAAAAAATTTTTTTTTTTTTTTTTT                              chr1
[2]    40 GGGGGGGGGGGGGGGGGGGGCCCCCCCCCCCCCCCCCC                              chr2
```

The genome has to be a named `DNAStringSet` or a filename that points to a FASTA-file saved somewhere on disk. By default, when omitting the genome parameter, `simulateSV` will load the human genome (hg19) automatically. This requires an installation of the R-package *BSgenome.Hsapiens.UCSC.hg19*. When using other *BSgenome* packages, it is recommended to extract the desired sequences first and combine them into a named `DNAStringSet`. For example, the preparation of the genome of an Ecoli strain (str. 536) would look like:

```
> library(BSgenome.Ecoli.NCBI.20080805)
> genome = DNAStringSet(Ecoli[["NC_008253"]])
> names(genome) = "NC_008253"
```

Each implemented SV will be reported with its position in the "normal" reference genome, and the breakpoint sequences. The rearranged genome is returned as `DNAStringSet` and the SV information is stored in its `metadata` slot as a named `list` of `data.frames`. All this can also be written to disk by specifying an `output` directory (which is the current directory by default). The SV tables are saved as CSV files (called deletions.csv, insertions.csv etc.) and the genome in FASTA format (genome_rearranged.fasta).

Note, that the seeds for the randomizations in the following examples were set for demonstration purposes only. The `seed` parameter can be omitted or used to reproduce the same simulation several times.
The parameter `output` is set to `NA` in all examples to avoid wrtiting the output to disc. The parameter `verbose` is set to `FALSE` to suppress progress information about the simulation (which is enabled by default).

2.1 Deletions

A segment is cut out from the genome. The following example generates three deletions of 10bp each:

```
> sim = simulateSV(output=NA, genome=genome, dels=3, sizeDels=10,
 bpSeqSize=6, seed=456, verbose=FALSE)
> sim

DNAStringSet object of length 2:
    width seq                                                                 names
[1]    20 AAAAAAAAAAAAAAAAAATT                                                chr1
[2]    30 GGGGGGGGGGGGGGGGGGGGCCCCCCCCCC                                      chr2

> metadata(sim)

$deletions
       Name  Chr Start End Size BpSeq
1 deletion1 chr1    21  30   10   ATT
2 deletion2 chr2    31  40   10   CCC
3 deletion3 chr1    19  28   10 AAATT
```

Chromosome 2, which harbours two deletions, is now 10bp shorter than chromosome 1. The breakpoint sequence of 6bp shows the 3bp up- and downstream of the deletion breakpoint in the rearranged genome.

2.2 Insertions

A segment is cut or copied from one chromosome A and inserted into another chromosome B. The following example generates three insertions of 5bp each:

```
> sim = simulateSV(output=NA, genome=genome, ins=3, sizeIns=5, bpSeqSize=6,
 seed=246, verbose=FALSE)
> sim

DNAStringSet object of length 2:
    width seq                                                    names
[1]    35 AAAAAAAAAAAAAAATTTTTTTTTTTTTTTTTTT                     chr1
[2]    45 GAAAAAGGGGGGGCCCCCGGGGGGGGGGGGCCCCCCCCCCCCCCC         chr2

> metadata(sim)

$insertions
         Name ChrA StartA EndA ChrB StartB EndB Size Copied BpSeqA BpSeqB_5prime
1 insertion_1 chr1     13   17 chr2      2    6    5  FALSE AAAAAA          GAAA
2 insertion_2 chr1     33   37 chr1     25   29    5  FALSE TTTTTT        TTTTTT
3 insertion_3 chr2     34   38 chr2      9   13    5  FALSE  CCCCC        GGGCCC
  BpSeqB_3prime
1        AAAGGG
2        TTTTTT
3        CCCGGG
```

Regarding insertion_1, for example, the 5bp segment `AAAAA` has been removed from chr1:14-18 and inserted into chr2:19-23. There are three breakpoint sequences reported for each insertion: the sequence at the deletion on chrA and at the 5' and 3' end of its insertion on chrB.

Setting the parameter `percCopiedIns` (range: 0-1, i.e. 0%-100%) can change the amount of "copy-and-paste-like" insertions.
The same example as before, with the difference that two of the three inserted sequences are copied:

```
> sim = simulateSV(output=NA, genome=genome, ins=3, sizeIns=5, percCopiedIns=0.66,
 bpSeqSize=6, seed=246, verbose=FALSE)
> sim

DNAStringSet object of length 2:
    width seq                                                    names
[1]    45 AAAAAAAAAAAAAAAAAAAATTTTTTTTTTTTTTTTTTTTTTT           chr1
[2]    45 GAAAAAGGGGGGGCCCCCGGGGGGGGGGGGCCCCCCCCCCCCCCC         chr2

> metadata(sim)

$insertions
         Name ChrA StartA EndA ChrB StartB EndB Size Copied BpSeqA BpSeqB_5prime
1 insertion_1 chr1     13   17 chr2      2    6    5   TRUE                GAAA
2 insertion_2 chr1     33   37 chr1     25   29    5   TRUE              TTTTTT
3 insertion_3 chr2     34   38 chr2      9   13    5  FALSE  CCCCC        GGGCCC
  BpSeqB_3prime
1        AAAGGG
2        TTTTTT
3        CCCGGG
```

The same sequence `AAAAA` from insertion_1 is now duplicated before insertion into chr2:19-23. Here, no breakpoint sequence is reported for the region on chr1, since this chromosome is not altered.

2.3 Inversions

A segment is cut from one chromosome and its reverse complement is inserted at the same place without loss or a shift of sequence. The example below assigns a different size for each inversion:

```
> sim = simulateSV(output=NA, genome=genome, invs=3, sizeInvs=c(2,4,6),
 bpSeqSize=6, seed=456, verbose=FALSE)
> sim

DNAStringSet object of length 2:
    width seq                                                  names
[1]    40 AAAAAAAAAAAAAAAAAAAAAATTTTTTTTTTTTTTTTT              chr1
[2]    40 GGGGGGGGGGGGGGGGGGGGGCCCCCCCCCGGGGCCCCCC             chr2

> metadata(sim)

$inversions
        Name  Chr Start End Size BpSeq_3prime BpSeq_5prime
1 inversion1 chr1    21  22    2       AAATTT       AAAAAT
2 inversion2 chr2    31  34    4       GGGCCC       CCCGGG
3 inversion3 chr2    19  24    6       GCCCCC       GGGGGG
```

Inversions have two breakpoint sequences, one for the 5' end and one for the 3' end of the inverted segment.

2.4 Tandem duplications

A segment is duplicated one after the other. The number of duplications is determined randomly. The parameter `maxDups` sets the maximum. The following example generates an, at most, tenfold tandem duplication of a 6bp sequence:

```
> sim = simulateSV(output=NA, genome=genome, dups=1, sizeDups=6, maxDups=3,
 bpSeqSize=6, seed=3456, verbose=FALSE)
> sim

DNAStringSet object of length 2:
    width seq                                                  names
[1]    40 AAAAAAAAAAAAAAAAAAAATTTTTTTTTTTTTTTTT              chr1
[2]    58 GGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGCCCCCCCCCCCCCCCCCC  chr2

> metadata(sim)

$tandemDuplications
                Name  Chr Start End Size Duplications  BpSeq
1 tandemDuplication1 chr2    12  17    6            3 GGGGGG
```

Here, the breakpoint sequence is the sequence at the end of one duplicated segment and the start of the following one. In this example the duplicated sequence is `AAATTT` and it has been repeated another two times.

2.5 Translocations

A segment from the 5' or 3' end of one chromosome A is exchanged with the 5' or 3' end of another chromosome B. If it is not balanced, the segment from chromosome B will be lost, what results in a duplicated sequence from chromosome A. The parameter `percBalancedTrans` sets the amount of balanced translocation (0-1, i.e. 0%-100%); by default, all translocations will be balanced. Segments which are translocated between two different ends (5'↔'3' of 3'↔5') are always inverted. After random generation of the breakpoint, the translocation spans the chromosome until the closest of both ends (which may include the centromere in the human genome).

```
> sim = simulateSV(output=NA, genome=genome,trans=1, bpSeqSize=6, seed=123, verbose=FALSE)
> sim

DNAStringSet object of length 2:
    width seq                                                   names
[1]    39 AAAAAAAAAAAAAAAAAAAATTTTTTTTTTTTTTTCCC                chr1
[2]    41 AAAAGGGGGGGGGGGGGGGGCCCCCCCCCCCCCCCCCCCCC             chr2

> metadata(sim)

$translocations
             Name ChrA StartA EndA SizeA ChrB StartB EndB SizeB Balanced BpSeqA BpSeqB
1 translocation_1 chr2      1    3     3 chr1     37   40     4     TRUE AAAGGG TTTCCC
```

This example exchanges the last 3bp of chromosome 2 with the first 19bp of chromosome 1. Both sequences were inverted.

The same example in an unbalanced fashion:

```
> sim = simulateSV(output=NA, genome=genome,trans=1, percBalancedTrans=0,
 bpSeqSize=6, seed=123, verbose=FALSE)
> sim

DNAStringSet object of length 2:
    width seq                                                   names
[1]    39 AAAAAAAAAAAAAAAAAAAATTTTTTTTTTTTTTTCCC                chr1
[2]    40 GGGGGGGGGGGGGGGGGGGGCCCCCCCCCCCCCCCCCCCC             chr2

> metadata(sim)

$translocations
             Name ChrA StartA EndA SizeA ChrB StartB EndB SizeB Balanced BpSeqA BpSeqB
1 translocation_1 chr2      1    3     3 chr1     37   40     4    FALSE        TTTCCC
```

The sequence from chr1:1-19 is lost, while there are now two copies of the translocated (inverted) segment from chr2:38-40. The breakpoint sequence for chr2 (denoted by chrA in general) is not reported, since this chromosome is not altered.

3 Simulation of biases towards SV formation mechanisms and repeat regions

By default, the SV breakpoints are placed uniformly across the genome. Several studies have shown, that structural variation formation in the human genome is not a random process but rather the

result of mechanisms such as nonallelic homologous recombination (NAHR), nonhomologous recombination (NHR), variable number of tandem repeats (VNTRs) and transposable element insertions (TEIs) ([Mills *et al.*, 2011], [Pang *et al.*, 2013]). These mechanisms can be further associated with repeat elements and regions of high homology such as LINEs, SINEs, Micro-/Minisatellites and segmental duplications ([Lam *et al.*, 2010]).
Using the default genome hg19 and setting `repeatBias` to `TRUE`, *RSVSim* simulates a bias of breakpoint positioning towards repeat regions. This is done in two steps:

1. Weighting SV formation mechanisms (here: NAHR, NHR, VNTR, TEI, Other) for each SV type. The type "Other" can be used for any individual weighting of repeats; by default, "Other" is associated with a random breakpoint.

2. Weighting each SV formation mechanism for each kind of repeat. The following types of repeat regions are supported: LINE/L1, LINE/L2, SINE/Alu, SINE/MIR, segmental duplications (SD), tandem repeats (TR; mainly micro-/minisatellites) and Random. The latter, "Random", means any region on the genome.

For the mechanism NAHR, both breakpoints will lie within a repeat region (with at least 50bp distance to the repeat margins), while for NHR, VNTR, TEI and Other, the repeat must make up for at least 75% of the SV region.
This feature is turned off automatically, when the user specifies his own genome (i.e. any genome other than hg19).
The default weights for SV mechanisms for deletions, insertions an duplications are based on figure 4b in [Mills *et al.*, 2011]. The weights for inversions refer to figure 3c in [Pang *et al.*, 2013]. The mechanisms and breakpoint sequences of translocations have not been studied as extensively as for other kinds of SVs. The default weights for translocations were chosen according to some exemplary publications ([Ou *et al.*, 2011], [Chen *et al.*, 2008]), so that NAHR, NHR and random breakpoint positioning contribute equally. In all cases, the results for SVs >1.000bp were used. The exact weights are:

```
       dels  ins invs dups trans
NAHR   0.23 0.10 0.65 0.10  0.33
NHR    0.69 0.03 0.35 0.03  0.33
TEI    0.04 0.82 0.00 0.82  0.00
VNTR   0.04 0.05 0.00 0.05  0.00
Other  0.00 0.00 0.00 0.00  0.34
```

The default weights for repeat regions for every SV mechanism were based on the enrichment analysis in [Lam *et al.*, 2010] (see their supplemental table 5). The exact values are:

```
> data(weightsRepeats, package="RSVSim")
> show(weightsRepeats)

       NAHR  NHR  TEI VNTR Other
L1     0.59 1.04 1.66    0     0
L2     0.13 0.62 0.25    0     0
Alu    2.72 1.16 0.47    0     0
MIR    0.14 0.74 0.14    0     0
SD     5.95 2.06 0.57    0     0
TR     0.00 0.00 0.00    1     0
Random 1.00 1.00 1.00    0     1
```

The user may provide other weights by passing his own `data.frames`, using the function arguments `weightsMechanisms` and `weightsRepeats`. The structure of the `data.frames` has to be identical to the default ones shown above (i.e. same dimensions, column and row names). The effect of the weights

is comparable to the `prob` argument in the R function `sample`.
For example would exclude tandem repeats, segmental duplications and random regions from the simulation (except for VNTRs) by setting their weights to zero for all mechanisms. NAHRs, would be related to SINEs only. For `weightsMechanisms`, the default values will be used, because the argument is missing here. Note, that `repeatBias=TRUE` has to be set to use this feature.

This feature requires the coordinates of repeat regions for hg19. This can be handled in two ways:

- *RSVSim* downloads the coordinates once automatically from the UCSC Browser's RepeatMasker track (which may take up to 45 Minutes!).
- The user may specify the filename of a RepeatMasker output file downloaded from their homepage ([Smit *et al.*, 1996-2010]): http://www.repeatmasker.org/species/homSap.html (e.g. hg19.fa.out.gz). Loading this file takes only a few minutes.

In both cases, RSVSim saves the coordinates as `RData` object `repeats_hg19.RData` to the RSVSim installation directory for a faster access in the future (if write privileges allow to do so). After that, one of the two steps mentioned above is not necessary anymore and next time, *RSVSim* is going to load the coordinates automatically from the `RData` file.
When loading the repeats, neighboured ones with a distance up to 50bp will be merged, to obtain larger repeat regions and to allow SVs to span more than one repeat. But, breakpoints will only be placed within repeats of the same type (e.g. LINE/L1-LINE/L1, or SINE/MIR-SINE/MIR etc.).

4 Simulation of additional breakpoint mutations

SV breakpoints are usually not clean but tend to co-occur with other, usually smaller mutations, such as indels or SNPs. *RSVSim* allows to randomly generate additional SNPs and indels within the flanking regions of each breakpoint. Their generation can be configured by the four arguments `bpFlankSize`, `percSNPs`, `indelProb` and `maxIndelSize`, which specify the size of the flanking regions in bp (i.e. proximity of the mutations to the breakpoint), the fraction on SNPs (in %), the probability of an indel (insertions and deletions are equally likely) and the maximum size of an indel (size is selected randomly between 1 and `maxIndelSize`). Each flanking region may only contain one indel. SNPs can affect 0-100% of the region. By default, this feature is turned off.
The following example creates one deletion with 10% SNPs and 100% indel probability within 10bp up-/downstream of the breakpoint:

```
> sim = simulateSV(output=NA, genome=genome, dels=1, sizeDels=5, bpFlankSize=10,
 percSNPs=0.25, indelProb=1, maxIndelSize=3, bpSeqSize=10, seed=123, verbose=FALSE)
> sim

DNAStringSet object of length 2:
    width seq                                                                names
[1]    40 AAAAAAAAAAAAAAAAAAAATTTTTTTTTTTTTTTTTT                            chr1
[2]    34 GGGGGGTGGGGGGGCCCCCCCCCCCCCCCCCCCC                                chr2

> metadata(sim)

$deletions
       Name  Chr Start End Size   BpSeq
1 deletion1 chr2     3   7    5 GGGGGGT
```

In addition to the 5bp deletion of the sequence `CCCCC`, two SNPs `C->T` and `C->A`, and a deletion of two more `C`s were added up- and downstream of the breakpoint.

5 Simulation within a genome subset

It is possible to run the simulation to certain chromosomes only by specifying the chromosome names in the parameter `chrs`. It has to be taken care that these chromosome names match the names in the `DNAStringSet` containing the genome sequences (e.g. "chr1", "chr2", ..., "chrY" for the default genome hg19 from the package *BSgenome.Hsapiens.UCSC.hg19*).
Furthermore, every SV has it's own parameter to restrict the simulation to a desired set of genomic regions: `regionsDels`, `regionsIns`, `regionsInvs`, `regionsDups` and `regionsTrans`. Each one being a `GRanges` object with a chromosome name, start- and end-position.
The following example places randomly four inversions into the second half of chr1 and the first half of chr2:

```
> regions = GRanges(IRanges(c(21,1),c(40,20)), seqnames=c("chr1","chr2"))
> regions

GRanges object with 2 ranges and 0 metadata columns:
      seqnames    ranges strand
         <Rle> <IRanges>  <Rle>
  [1]     chr1     21-40      *
  [2]     chr2      1-20      *
  -------
  seqinfo: 2 sequences from an unspecified genome; no seqlengths

> sim = simulateSV(output=NA, genome=genome, invs=4, sizeInvs=5,
 regionsInvs=regions, bpSeqSize=6, seed=2345, verbose=FALSE)
> sim

DNAStringSet object of length 2:
    width seq                                                                names
[1]    40 AAAAAAAAAAAAAAAAAAAATAAAAATAAAAATTTTTTTT                           chr1
[2]    40 CCCCCCCCCCGGGGGGGGGGCCCCCCCCCCCCCCCCCCCC                           chr2

> metadata(sim)

$inversions
        Name  Chr Start End Size BpSeq_3prime BpSeq_5prime
1 inversion1 chr1    22  26    5       AAATAA       AATAAA
2 inversion2 chr1    28  32    5       AAATTT       AATAAA
3 inversion3 chr2     6  10    5       CCCGGG       CCCCCC
4 inversion4 chr2     1   5    5       CCCCCC
```

For translocations, the regions only say where to place the breakpoint, since the translocated region spans the chromosome until the closest of both ends.

Some applications may focus on certain parts of the hg19 only, like exons, introns or transcripts. The package *GenomicFeatures* provides functionalities to export such coordinates from the UCSC Genome Browser to R (see for example `makeTxDbFromUCSC`, `exonsBy`, `intronsBy`, `transcriptsBy`). In the following example, 100 deletions would be placed somewhere in the exonic regions on hg19:

```
> transcriptDB = makeTxDbFromUCSC(genome = "hg19",tablename = "knownGene")
> exons = exonsBy(transcriptDB)
> exons = unlist(exons)
> exons = GRanges(IRanges(start=start(exons), end=end(exons)), seqnames=seqnames(exons))
> simulateSV(output=NA, dels=100, regionsDels=exons, sizeDels=1000, bpSeqSize=50)
```

SVs will not be placed within unknown regions or assembly gaps denoted by the letter `N`. Such regions are detected and filtered automatically.

5.1 Inserting a set of SVs

The simulation allows to turn off the random generation of breakpoints and to insert a set of (for example previously detected or known) SVs. It works by using the same regions parameters and setting the parameter `random` to `FALSE`. This may also be a vector of five `TRUE`/`FALSE` values (in the order: deletions, insertions, inversions, tandem duplications, translocations) if some SVs shall be generated randomly and others not.
The example below inserts a deletion at chr2:16-25:

```
> knownDeletion = GRanges(IRanges(16,25), seqnames="chr2")
> names(knownDeletion) = "myDeletion"
> knownDeletion

GRanges object with 1 range and 0 metadata columns:
             seqnames    ranges strand
                <Rle> <IRanges>  <Rle>
  myDeletion     chr2     16-25      *
  -------
  seqinfo: 1 sequence from an unspecified genome; no seqlengths

> sim = simulateSV(output=NA, genome=genome, regionsDels=knownDeletion,
 bpSeqSize=10, random=FALSE, verbose=FALSE)
> sim

DNAStringSet object of length 2:
    width seq                                                          names
[1]    40 AAAAAAAAAAAAAAAAAAAATTTTTTTTTTTTTTTTTT                       chr1
[2]    30 GGGGGGGGGGGGGGGCCCCCCCCCCCCCCC                               chr2

> metadata(sim)

$deletions
                 Name  Chr Start End Size      BpSeq
myDeletion myDeletion chr2    16  25   10 GGGGGCCCCC
```

Note, that the output adopts the names, that were given the `GRanges` object of the inserted SV(s).

It's a little different for insertions and translocations, since they involve two genomic regions. Thus, the `GRanges` object for `regionsIns` has to be extended by columns `chrB` and `startB`, saying, that the sequence within ranges of the `GRanges` object will be inserted at chrB:startB.
The next example inserts the sequence from chr1:16:25 at chr2:26:

```
> knownInsertion = GRanges(IRanges(16,25),seqnames="chr1", chrB="chr2", startB=26)
> names(knownInsertion) = "myInsertion"
> knownInsertion

GRanges object with 1 range and 2 metadata columns:
              seqnames    ranges strand |        chrB    startB
                 <Rle> <IRanges>  <Rle> | <character> <numeric>
  myInsertion     chr1     16-25      * |        chr2        26
  -------
  seqinfo: 1 sequence from an unspecified genome; no seqlengths

> sim = simulateSV(output=NA, genome=genome, regionsIns=knownInsertion,
 bpSeqSize=10, random=FALSE, verbose=FALSE)
> sim
```

```
DNAStringSet object of length 2:
    width seq                                                   names
[1]    30 AAAAAAAAAAAAAAATTTTTTTTTTTTTT                        chr1
[2]    50 GGGGGGGGGGGGGGGGGGGGCCCCCAAAAATTTTTCCCCCCCCCCCCCCC   chr2

> metadata(sim)

$insertions
                   Name ChrA StartA EndA ChrB StartB EndB Size Copied     BpSeqA
myInsertion myInsertion chr1     16   25 chr2     26   35   10  FALSE AAAAATTTTT
            BpSeqB_5prime BpSeqB_3prime
myInsertion    CCCCCAAAAA    TTTTTCCCCC
```

The `GRanges` object for translocations has to be extended by columns `chrB`, `startB` and `endB`, saying, the sequence within the ranges of the object will be exchanged with the sequence from chrB:startB-endB. Typically, one start/end of each region equals the 5'or 3' end of the chromosome. One may add a column `balanced` saying `TRUE/FALSE` for every single entry.
The next example is a simple simulation of the translocation t(9;22) leading to the BCR-ABL fusion gene. It uses simple breakpoints within 9q34.1 and 22q11.2 for demonstration:

```
> trans_BCR_ABL = GRanges(IRanges(133000000,141213431), seqnames="chr9",
 chrB="chr22", startB=23000000, endB=51304566, balanced=TRUE)
> names(trans_BCR_ABL) = "BCR_ABL"
> trans_BCR_ABL

GRanges object with 1 range and 4 metadata columns:
          seqnames              ranges strand |        chrB    startB      endB  balanced
             <Rle>           <IRanges>  <Rle> | <character> <numeric> <numeric> <logical>
  BCR_ABL     chr9 133000000-141213431      * |       chr22  23000000  51304566      TRUE
  -------
  seqinfo: 1 sequence from an unspecified genome; no seqlengths

> sim = simulateSV(output=NA, chrs=c("chr9", "chr22"), regionsTrans=trans_BCR_ABL,
 bpSeqSize=50, random=FALSE)
```

The example is not executed here, because it requires the package *BSgenome.Hsapiens.UCSC.hg19*. Setting the argument `transInsert=20` adds up to 20 random nucleotides at both breakpoints.

It is strongly recommended to only use a set of non-overlapping SVs.

6 Comparing two sets of SVs

A typical use case of SV simulation with is the evaluation of SV detection algorithms. The function `compareSV` looks for overlaps between the output of the simulation, the ground truth (`simSVs`), and the output of an SV detection program (`querySVs`) up to a certain tolerance. It computes the sensitivity, precision and the percentage overlap between the breakpoint sequences (if available).
An overlap is defined as the overlap between the breakpoints/breakpoint regions in `simSVs`/`querySVs` up to the given tolerance in bp. Overlap does not mean the whole affected region between the start and end of the SV.
Unfortunately, there is currently no common standard format for SVs. Because the main information about SVs is their position in the genome and, sometimes, the breakpoint sequence (which depends on the SV detection algorithm), `compareSV` expects the SV detections as tables in a simple BED- or

BEDPE format (`http://code.google.com/p/bedtools`). Deletions, inversions and tandem duplications, which affect one region on the genome, can be either given in both formats. Translocations and insertions, which affect to regions on the genome, require the BEDPE-format. Eventually, the output of the SV detection format has to be converted accordingly (for example in R).
The function only compares one SV type at a time, so `querySVs` and `simSVs` may not contain a mixture of different kinds of SVs.
If the BED-tables for `querySVs` or the simulation output are saved on disk, `compareSV` also accepts their filenames and loads the tables automatically as `data.frame` in R.

The following example simulates first five 5bp deletions in the small toy genome defined above:

```
> sim = simulateSV(output=NA, genome=genome, dels=5, sizeDels=5,
 bpSeqSize=10, seed=2345, verbose=FALSE)
> simSVs = metadata(sim)$deletions
> simSVs
```

```
       Name  Chr Start End Size      BpSeq
1 deletion1 chr1    34  38    5    TTTTTTT
2 deletion2 chr1     2   6    5     AAAAAA
3 deletion3 chr2    22  26    5 GGGGCCCCCC
4 deletion4 chr2    31  35    5 CCCCCCCCCC
5 deletion5 chr1    12  16    5 AAAAAAAAAT
```

An SV detection in BED format (the `querySVs`) may look like this: Four of five deletions were detected, two with exact and two with an approximate breakpoint. Two additional deletions were detected, which were not part of the simulation.

```
> querySVs = data.frame(
   chr=c("chr1","chr1","chr1","chr2","chr2"),
   start=c(12,17,32,2,16),
   end=c(15,24,36,6,20),
   bpSeq=c("AAAAAAAAAA", "AAAAAAATTT", "TTTTTTTTTT",
     "GGGGGGGGGG", "GGGGGGCCCC")
 )
> querySVs
```

```
   chr start end      bpSeq
1 chr1    12  15 AAAAAAAAAA
2 chr1    17  24 AAAAAAATTT
3 chr1    32  36 TTTTTTTTTT
4 chr2     2   6 GGGGGGGGGG
5 chr2    16  20 GGGGGGCCCC
```

The column with the breakpoint sequence is optional, the column names not important (BED-files have no header).
A comparison with 0bp tolerance yields only two overlaps:

```
> compareSV(querySVs, simSVs, tol=0)
```

```
       Name  Chr Start End Size      BpSeq Overlap OverlapBpSeq
1 deletion1 chr1    34  38    5    TTTTTTT                   NA
2 deletion2 chr1     2   6    5     AAAAAA                   NA
```

```
3 deletion3 chr2    22  26    5 GGGGCCCCCC                    NA
4 deletion4 chr2    31  35    5 CCCCCCCCCC                    NA
5 deletion5 chr1    12  16    5 AAAAAAAAAT                    NA
```

A higher breakpoint tolerance of +/- 3bp also includes more imprecise detections:

```
> compareSV(querySVs, simSVs, tol=3)

       Name  Chr Start End Size      BpSeq    Overlap OverlapBpSeq
1 deletion1 chr1    34  38    5    TTTTTTT chr1:32-36          100
2 deletion2 chr1     2   6    5     AAAAAA                      NA
3 deletion3 chr2    22  26    5 GGGGCCCCCC                      NA
4 deletion4 chr2    31  35    5 CCCCCCCCCC                      NA
5 deletion5 chr1    12  16    5 AAAAAAAAAT chr1:12-15           90
```

Note that for `deletion1`, the breakpoint sequence matched only by 80%.

The second example compares translocations:

```
> sim = simulateSV(output=NA, genome=genome, trans=2, percBalancedTrans=0.5,
 bpSeqSize=10, seed=246, verbose=FALSE)
> simSVs = metadata(sim)$translocations
> simSVs

             Name ChrA StartA EndA SizeA ChrB StartB EndB SizeB Balanced     BpSeqA
1 translocation_1 chr1      1   13    13 chr2      1    2     2    FALSE
2 translocation_2 chr1     37   40     4 chr2     29   40    12     TRUE TTTTTCCCCC
      BpSeqB
1 AAAAAGGGGG
2  CCCCCTTTT
```

Detected translocations have to be given in BEDPE-format (i.e. at least six columns chr1, start1, end1, chr2, start2, end2 for the breakpoints on both chromosomes). In this example, the breakpoints were approximated up to 1bp or 2bp, optional breakpoint sequences are missing:

```
> querySVs = data.frame(
   chr=c("chr1", "chr1", "chr2"),
   start1=c(15,32,32),
   end1=c(18,36,33),
   chr2=c("chr2","chr2","chr1"),
   start2=c(10,31,32),
   end2=c(12,33,36)
 )
> querySVs

   chr start1 end1 chr2 start2 end2
1 chr1     15   18 chr2     10   12
2 chr1     32   36 chr2     31   33
3 chr2     32   33 chr1     32   36
```

Here, all detected SVs span the simulated breakpoints:

```
> compareSV(querySVs, simSVs, tol=0)
```

```
             Name ChrA StartA EndA SizeA ChrB StartB EndB SizeB Balanced     BpSeqA
1 translocation_1 chr1      1   13    13 chr2      1    2     2    FALSE
2 translocation_2 chr1     37   40     4 chr2     29   40    12     TRUE TTTTTCCCCC
      BpSeqB Overlap OverlapBpSeq_A OverlapBpSeq_B
1 AAAAAGGGGG                    NA             NA
2  CCCCCTTTT                    NA             NA
```

7 Setting structural variation sizes

One may specify just one size in the parameters `sizeDels`, `sizeIns`, `sizeInvs` or `sizeDups` that applies to every SV of each type. But often, it might be more realistic to assign an individual, arbitrary size to every single SV. In the simplest case, they may be uniformly distributed:

```
> sizes = sample(2:5, 5, replace=TRUE)
> sizes

[1] 4 4 4 2 2

> sim = simulateSV(output=NA, genome=genome, dels=5, sizeDels=sizes,
 bpSeqSize=6, seed=246, verbose=FALSE)
> sim

DNAStringSet object of length 2:
    width seq                                                 names
[1]    30 AAAAAAAAAAAAAATTTTTTTTTTTTTT                        chr1
[2]    34 GGGGGGGGGGGGGGGGCCCCCCCCCCCCCCCCCC                  chr2

> metadata(sim)

$deletions
       Name  Chr Start End Size  BpSeq
1 deletion1 chr1    13  16    4 AAAAAA
2 deletion2 chr2     2   5    4   GGGG
3 deletion3 chr1    32  35    4 TTTTTT
4 deletion4 chr1     8   9    2 AAAAAA
5 deletion5 chr2    29  30    2 CCCCCC
```

7.1 Estimating size distribution from real data

According to studies from the 1000 Genomes Project, for deletions, insertions and duplications, the amount of SVs decreases rather quickly as their size increases ([Mills *et al.*, 2011]). The function `estimateSVSizes` simulates SV sizes by fitting a beta distribution, which is flexible enough to realistically model the shape of the size distribution of all four SV types. Its two shape parameters can be derived from a given vector of SV sizes. This requires the R-package *MASS*.
The following toy example draws 1.000 SV sizes between 10bp and 1000bp from a beta distribution based on a vector of 15 SV sizes:

```
> svSizes = c(10,20,30,40,60,80,100,150,200,250,300,400,500,750,1000)
> simSizes = estimateSVSizes(n=1000, svSizes=svSizes, minSize=10, maxSize=1000, hist=TRUE)
> head(simSizes, n=20)

 [1] 426 343  39  27 275 971  48  21 998  27 143  10 571  16 716 937 865  10 882  11
```

The `minSize` and `maxSize` can be omitted; they are then calculated from the given set of `svSizes`. It is recommended to uses a `minSize` and `maxSize` that is consistent with the minimum/maximum values in `svSizes`.
Setting the parameter `hist=TRUE` also plots a histogram of the SV sizes to give an impression of their distribution (see Fig.1).

For deletions, insertions, inversions and tandem duplications, `estimateSVSizes` can use default parameters for the beta distribution. They were estimated from the Database of Genomic Variants (DGV) release 2012-03-29 ([Iafrate *et al.*, 2004]). Hence, no set of SV sizes is needed for fitting the distribution. In total, 1.129 deletions, 490 insertions, 202 inversions and 145 tandem duplications between 500bp and 10kb were used to estimate the shape. The parameter `default` can be set to either `"deletions"`, `"insertions"`, `"inversions"` or `"tandemDuplications"` to use the according set of shape parameters:

```
> delSizes = estimateSVSizes(n=10000, minSize=500, maxSize=10000,
 default="deletions", hist=TRUE)
> head(delSizes, n=15)

 [1] 1838  503 2240 1071  853  751 1147  671  629  776 1814 1773  517  502  581

> delSizes = estimateSVSizes(n=10000, minSize=500, maxSize=10000,
 default="insertions", hist=TRUE)
> head(delSizes, n=15)

 [1]  833 1033 3816 1334 1723 2048 2521 2874 1877  509 6090  743 1387 1796 5053

> invSizes = estimateSVSizes(n=10000, minSize=500, maxSize=10000,
 default="inversions", hist=TRUE)
> head(invSizes, n=15)

 [1] 2976 1769 6532 1991 1083 1678 2841 1886 4044 1037  649 2563  771 3487 1213

> delSizes = estimateSVSizes(n=10000, minSize=500, maxSize=10000,
 default="tandemDuplications", hist=TRUE)
> head(delSizes, n=15)

 [1] 6742 1260  762 5371 2623 1483 1593 1801 1388  559 1107 3039  649 2442  755
```

See Fig.2, Fig.3, Fig.4 and Fig.5 to see the estimated distribution based on the SVs in the DGV. When using these default values, it is recommended to simulate SVs that do not differ too much in size (aroung 500bp-10kb).

8 Runtime

The runtime of *RSVSim* mainly depends on the number of simulated breakpoints and the size of the genome.
The following test case simulates 50 SVs (10 per SV type) on the complete hg19:

```
> simulateSV(output=NA, dels=10, ins=10, inv=10, dups=10, trans=10,
 sizeDels=10000, sizeIns=10000, sizeInvs=10000, sizeDups=10000,
 repeatBias=FALSE, bpFlankSize=50, percSNPs=0.25, indelProb=0.5, maxIndelSize=10)
```

Ten repetitions of the simulation yield an average time of 6 minutes on a single Intel Xeon CPU with 2.40GHz an R version 2.15.2 (2012-10-26) including loading of the hg19 and writing of the output to

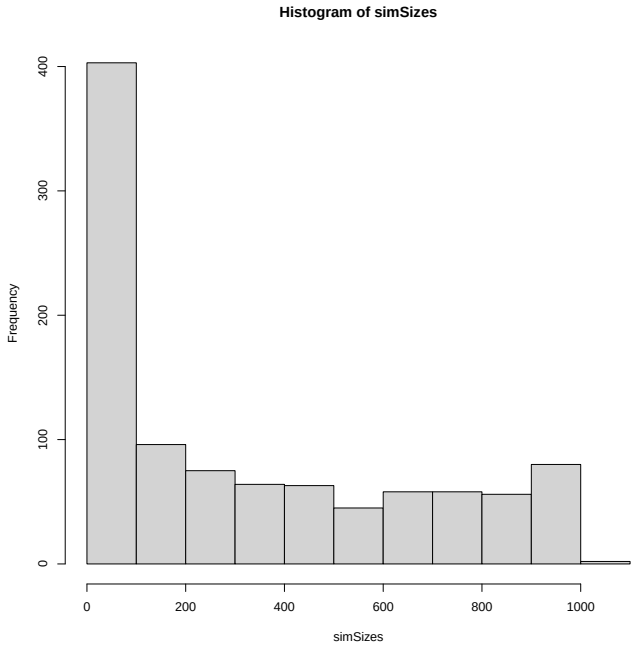

Figure 1: Distribution of 1.000 SV sizes drawn from a beta distribution using function `estimateSVSizes`.

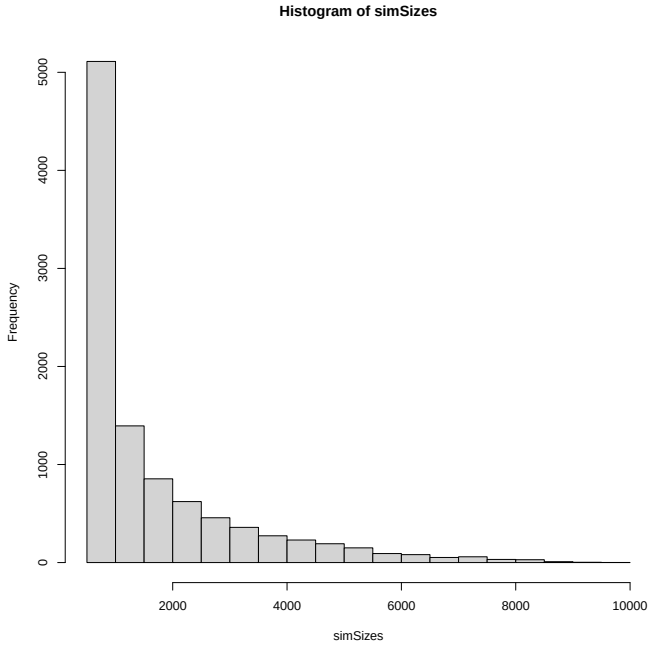

Figure 2: Distribution of 10.000 deletion sizes based on deletions from the Database of Genomic Variants.

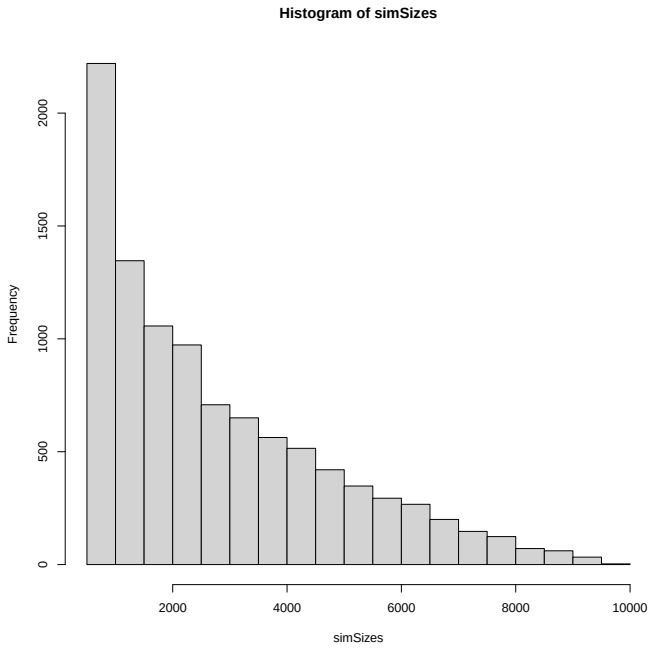

Figure 3: Distribution of 10.000 insertion sizes based on insertions from the Database of Genomic Variants.

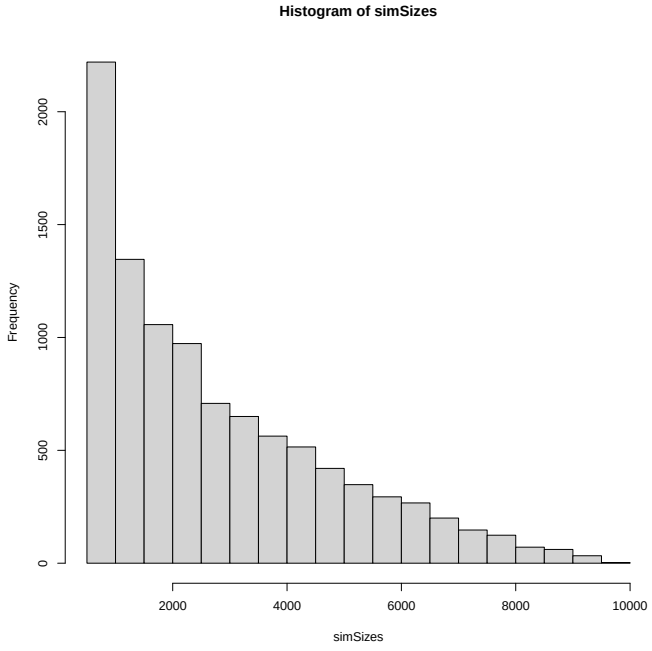

Figure 4: Distribution of 10.000 inversion sizes based on inversions from the Database of Genomic Variants.

disc.
Enabling biases for repeat regions for the same test case (i.e. `repeatBias=TRUE`), yield an average 7 minutes.
For other simulations on hg19, the runtime will scale linearly with the number of SV breakpoints.
Note, that the one-time, initial download of the repeat coordinates for hg19 from the UCSC browser may take up to 45 minutes. Alternatively, providing a RepeatMasker output file is much quicker (see section 3 for more details).

References

[Chen *et al.*, 2008] Chen W. *et al* (2008) Mapping translocation breakpoints by next-generation sequencing, *Genome Res*, **18(7)**, 1143-1149.

[Huang *et al.*, 2011] Huang W. *et al* (2011) ART: a next-generation sequencing read simulator, *Bioinformatics*, **28 (4)**, 593-594.

[Hu *et al.*, 2012] Hu X. *et al* (2012) pIRS: Profile-based Illumina pair-end reads simulator, *Bioinformatics*, **28(11)**, 1533-1535.

[Iafrate *et al.*, 2004] Iafrate A.J. *et al* (2004) Detection of large-scale variation in the human genome, *Nat Genet.*, **36(9)**, 949-951.

[Lam *et al.*, 2010] Lam H.Y. *et al* (2010) Nucleotide-resolution analysis of structural variants using BreakSeq and a breakpoint library, *Nat Biotechnol*, **28(1)**, 47-55.

[Mills *et al.*, 2011] Mills R.E. *et al* (2011) Mapping copy number variation by population-scale genome sequencing, *Nature*, **470(7332)**, 59-65.

[Ou *et al.*, 2011] Ou Z. *et al* (2011) Observation and prediction of recurrent human translocations mediated by NAHR between nonhomologous chromosomes, *Genome Res*, **21(1)**, 33-46.

[Pang *et al.*, 2013] Pang A.W. *et al* (2013) Mechanisms of Formation of Structural Variation in a Fully Sequenced Human Genome, *Hum Mutat*, **34(2)**, 345-354.

[Smit *et al.*, 1996-2010] Smit A. *et al* (1996-2010) RepeatMasker Open-3.0., <http://www.repeatmasker.org>.

9 Session Information

```
R version 4.0.0 (2020-04-24)
Platform: x86_64-apple-darwin17.0 (64-bit)
Running under: macOS Mojave 10.14.6

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRlapack.dylib

locale:
[1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

attached base packages:
[1] stats4    parallel  stats     graphics  grDevices utils     datasets  methods
[9] base
```

```
other attached packages:
[1] MASS_7.3-51.6        RSVSim_1.28.0        GenomicRanges_1.40.0 GenomeInfoDb_1.24.0
[5] Biostrings_2.56.0    XVector_0.28.0       IRanges_2.22.0       S4Vectors_0.26.0
[9] BiocGenerics_0.34.0

loaded via a namespace (and not attached):
 [1] zlibbioc_1.34.0             GenomicAlignments_1.24.0    BiocParallel_1.22.0
 [4] lattice_0.20-41             jpeg_0.1-8.1                hwriter_1.3.2
 [7] tools_4.0.0                 SummarizedExperiment_1.18.0 grid_4.0.0
[10] Biobase_2.48.0              png_0.1-7                   latticeExtra_0.6-29
[13] matrixStats_0.56.0          crayon_1.3.4                Matrix_1.2-18
[16] GenomeInfoDbData_1.2.3      RColorBrewer_1.1-2          bitops_1.0-6
[19] RCurl_1.98-1.2              DelayedArray_0.14.0         compiler_4.0.0
[22] Rsamtools_2.4.0             ShortRead_1.46.0
```

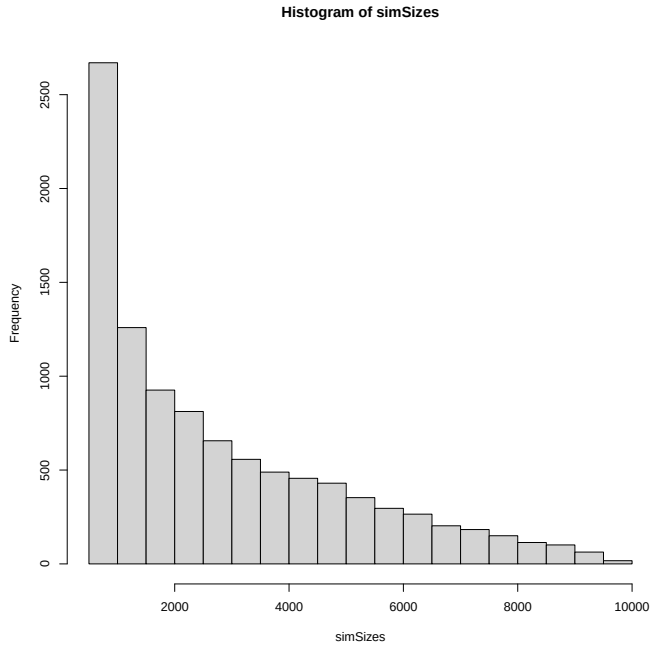

Figure 5: Distribution of 10.000 tandem duplication sizes based on tandem duplications from the Database of Genomic Variants.