Pysim-sv User Manual

Introduction

Pysim-sv is a bioinformatics tool written by Python for the simulation of germline and somatic structural variations (SVs) such as deletions, insertions, tandem duplications, inversions, translocations and copy number variations (CNVs). Pysim-sv can simulate cancer genomes with different features including purity, aneuploidy and heterogeneity. It can also generate Next-Generation Sequencing (NGS) reads according to the different cancer genomes, which can be mixed with specified proportions. GC bias can also be introduced to make the data conform better with the real situation.

Workflow

Pysim-sv is composed of 5 steps: specified ploidy for different genomes; insert SVs, CNVs and SNPs; generate NGS reads by ART; mix different clones and introduce GC bias. Each step is executed by a Python script. Most parameters are self-explainable. Generally, default settings are suitable for human 100bp reads. More details of parameters are provided below.

1. specify_ploidy.py

Description: generate the ploidy of genome according to the input config file.

Usage: python specify_ploidy.py –i <reference.fa> -c <config> -o <output_genome.fa>

Options:

-i FILE the reference fasta file such as hg19.fa

-c FILE a config file which specify the number of ploidy

-o FILE the output fasta format of genome file, default output_ploidy.fa

The <configFile> has the following format

#chromName	ploidy
chr1	2
chr2	3

In the <configFile>, the columns should be tab-delimited. The first row of this file is

assumed to be the header of the file. The first column (chromName) is the chromosome name. The second column (ploidy) is the number of specified ploidy, which should be 2 when simulating diploid genomes. If users take the reference genome as input, this step can be omitted.

2. simulate_SV.py

Description: simulate germline and somatic SVs and SNPs.

Usage: pysim_main.py –c ini.config

Options:

-h show this help message and exit

-c an initial config file which contains the parameters

The ini_config_file has the following format

```
[pysim_settings]
SV_config_file = /data1/XiLab/ycxia/simSV/config    ### read the file which contain the message of simulation SVs
ref_fasta=/data1/XiLab/ycxia/simSV/test_chr22.fa      ### the reference file
dbsnp=/data/qfxing/sim/snp/dbsnp_138.hg19.vcf         ### the vcf file which contains the SNP positions
somatic=Y      ### generate the somatic SVs, otherwise generate germline SVs merely
germline_num=100 ### the number of germline SNPs and indels
somatic_num=100 ### the number of somatic SNPs and indels
db=N ### the somatic SNPs is generated randomly
somatic_SNP_db='' ### if the parameter 'db=Y', setting the path of database which is written by vcf file;
hyp_rate=0.5    ### the proportion of heterozygous and homozygous;
up_down_stream=50 ### Size of the each breakpoint's flanking regions, which may contain additional SNPs and/or indels;
snp_rate=0.1### Percentage of SNPs within a breakpoint's flanking region;
indel_prob=1 ### Probability for an indel within a breakpoint's flanking region;
min_indel_length=5      ###Mimum size of an indel;
```

```
max_indel_length=15    ###Maximum size of an indel;

out_prefix=test_    ###Output prefiex for the rearranged genome and SV lists;

sub_clone=3 ### the number of subclones;

germ_ratio=0 ###the ratio of germline SV in somatic SV,
```

The <SV_config_File> has the following format

#type of SVs	Length of SVs	Number of SVs
del	1000	5
del	5000	10
inv	7000	5

In the <configFile>, the columns should be tab-delimited. The first row of this file is assumed to be the header of the file. The 1st column (type of SVs) is the type of SVs including del, inv, translocation, tandem duplication The 2nd column (Length of SVs) is the length of SVs. And the 3rd column (Number of SVs) is the number of this kind of SV.

3 run_art.py

Description: use ART to generate illumina paired-end reads.

Usage: run_art.py -i <file> -l <read length> -f <coverage> -m <insert size> -s <sd of insert size> -o <prefix_out_fastq>

Options:

-h	show this help message and exit
-i FILE	A fasta file which contains SVs and CNVs.
-l INT	paired-end read length, default 100
-f FLOAT	the coverage of reads,default 30
-m INT	library insert size, default 350
-s FLOAT	Standard Deviation of insert size, default 50
-o file	the prefix of output file, default output_

4. mixture_v0.5.py

Description: mix the paired-end reads which come from different genome with specified rates.

Usage: python mixture_v0.5.py –i <config file> -o <prefix of output fastq file>

Options:

-h	show this help message and exit
-i FILE	the specified rates of different genome reads
-o file	the prefix of paired end fastq format file

The <configFile> has the following format

#The path of the generated fasta genome	purity
sim_SV_genome_subclone1.fa	0.6
sim_SV_genome_subclone2.fa	0.2
normal.fa	0.2

In the <configFile>, the columns should be tab-delimited. The first row of this file is assumed to be the header of the file. The 1st column (The path of the generated fasta genome) is the path of the fasta forma genome. The 2nd column (purity) is the proportion of each subclone.

5. GC_bias.py

Description: Pysim-sv first calculates a subsampling probability for every read pair (or every read for single-end data). The probability depends on the local GC-content of the sequence from which the read pair is generated. Users can specify the function of this dependency. Pysim-sv then generates a random number from the Bernoulli distribution with the success rate as the calculated probability. Pysim-sv will select this read pair if the random number is 1, and will not if the number is 0.

Usage: python GC_bias.py -i <file> -g <GC_file> -b <name_sort_sam file> -o <out_fastq> -m <mode> -k <y/n>

Options:

-h show this help message and exit

-i FILE a reference fasta file.

-g FILE GC_function file, the filst col is GC content,this file can be generated by the default function, if the user choose the –m 1,2,or 3

-b FILE input name sorted sam/bam format file generated by ART

-o file prefix of output fastq format paired-end file

-m INT the mode of different function, 1 represents a linear function, 2 represents a quadratic function, 3 represents cubic function, 0 represents a function specified by user

-k y/n whether to generate sam format file, y represents generating sam format file, default n

Examples of working with Pysim-sv

1. To simulate germline SVs

In this simulation, we specify the reference (hg19) as input. We insert 10000 germline SNPs and indels, and 0 somatic SNPs. We use dpSNP 138 as input.

Step1:

python simulate_SV.py –c ini.config

The genome file name which contains germline SVs is sim_SV_genome_subclone1.fa

Step2:

python run_art.py -i sim_SV_genome_subclone1.fa -l 100 -f 30 -m 350 -s 50 -o sim_SV_genome_

The output would be sim_SV_genome_.sam, sim_SV_genome_1.fq and sim_SV_genome_2.fq

Step3:

python GC_bias.py -i sim_SV_genome_subclone1.fa -b sim_SV_genome_.sam -o sim_SV_genome -m 2 -k n

The output file would be sim_SV_genome_1.fq and sim_SV_genome_2.fq

2. To simulate Somatic SVs

In this simulation, we specify the reference (hg19) as input. We set both the number of germline and somatic SNPs and indels as 10000, specifically. We use dpSNP 138 as input.

Step1:

python simulate_SV.py –c ini.config

The genome file name which contains germline SVs is sim_SV_genome_subclone1.fa, and normal.fa

Step2:

python run_art.py -i sim_SV_genome_subclone1.fa -l 100 -f 30 -m 350 -s 50 -o sim_SV_genome_subclone1_

python run_art.py -i normal.fa -l 100 -f 30 -m 350 -s 50 -o normal_

The output would be sim_SV_genome_.sam, sim_SV_genome_1.fq and sim_SV_genome_2.fq, normal_.sam,normal_1.fq and normal_2.fq

Step3:

python GC_bias.py -i sim_SV_genome_subclone1.fa -b sim_SV_genome_.sam -o sim_SV_genome -m 2 -k n

python GC_bias.py -i normal.fa -b normal_.sam -o normal_genome -m 2 -k n

The output file would be sim_SV_genome_1.fq and sim_SV_genome_2.fq, normal_genome_1.fq and normal_genome_2.fq

3. To simulate Somatic SVs and tumor purity

In this simulation, the settings is the same as above. Two tumor clones are simulated independently. We generate the sequencing data of a tumor with 80% (tumor 1 with 60% and tumor 2 with 20%) purity.

Step1:

python simulate_SV.py –c ini.config

Step2:

python run_art.py -i sim_SV_genome_subclone1.fa -l 100 -f 30 -m 350 -s 50 -o sim_SV_genome_subclone1_

python run_art.py -i sim_SV_genome_subclone2.fa -l 100 -f 30 -m 350 -s 50 -o sim_SV_genome_subclone2_

python run_art.py -i normal.fa -l 100 -f 30 -m 350 -s 50 -o normal_

The output would be sim_SV_genome_subclone1_.sam, sim_SV_genome_subclone1_1.fq and sim_SV_genome_subclone1_2.fq, sim_SV_genome_subclone2.sam, sim_SV_genome_subclone2_1.fq and sim_SV_genome_subclone2_2.fq, normal_.sam, normal_1.fq and normal__2.fq

Step3:

python mixture_v0.4.py –i config -o mixture_tumor
The <configFile> has the following format

#The path of the generated fasta genome	purity
sim_SV_genome_subclone1.fa	0.6
sim_SV_genome_subclone2.fa	0.2
normal.fa	0.2

The output would be mixture_tumor_1.fq and mixture_tumor_2.fq

Step4:

sh bwa_map.sh mixture_tumor hg19.fa < input_fastq_dic > <output_dic >

The output would be mixture_tumor.sam

Step5:
python GC_bias.py -i hg19.fa -b mixture_tumor.sam -o sim_SV_genome -m 2 -k n
The output file would be sim_SV_genome_1.fq and sim_SV_genome_2.fq

4. To simulate tumor heterogeneity
In this simulation, the settings are the same as above. One tumor clone is simulated firstly. Then we use the tumor clone as input and simulate two subclones. We generate the sequencing data of a tumor with 90% (tumor 1 with 30% and tumor 2 with 30%,tumor 3 with 10%) purity.
Step1:
python simulate_SV.py –c ini.config
Step2:
python simulate_SV.py –c ini.config
The genome file name which contains germline SVs is
sim_SV_genome_subclone1_subclone1.fa, and
sim_SV_genome_subclone1_subclone2.fa
Step3:
python run_art.py -i sim_SV_genome_subclone1.fa -l 100 -f 30 -m 350 -s 50 -o sim_SV_genome_subclone1_
python run_art.py -i sim_SV_genome_subclone1_subclone1.fa -l 100 -f 30 -m 350 -s 50 -o sim_SV_genome_subclone1_subclone1_
python run_art.py -i sim_SV_genome_subclone1_subclone1.fa -l 100 -f 30 -m 350 -s 50 -o sim_SV_genome_subclone1_subclone2_
python run_art.py -i normal.fa -l 100 -f 30 -m 350 -s 50 -o normal_

The output would be sim_SV_genome_subclone1_.sam,
sim_SV_genome_subclone1_1.fq and sim_SV_genome_subclone1_2.fq,
sim_SV_genome_subclone1_subclone1.sam, sim_SV_genome_
subclone1_subclone1_1.fq and sim_SV_genome_subclone1_subclone1_2.fq
sim_SV_genome_subclone1_subclone2.sam, sim_SV_genome_

subclone1_subclone2_1.fq and sim_SV_genome_subclone1_subclone2_2.fq, normal_.sam, normal_1.fq and normal_2.fq

Step4:

python mixture_v0.4.py –i config -o mixture_tumor

The <configFile> has the following format

#The path of the generated fasta genome	purity
sim_SV_genome_subclone1.fa	0.3
sim_SV_genome_subclone1_subclone1.fa	0.3
sim_SV_genome_subclone1_subclone2.fa	0.3
normal.fa	0.1

The output would be mixture_tumor_1.fq and mixture_tumor_2.fq

Step5:

sh bwa_map.sh mixture_tumor hg19.fa < input_fastq_dic > <output_dic >

The output would be mixture_tumor.sam

Step6:

python GC_bias.py -i hg19.fa -b mixture_tumor.sam -o sim_SV_genome -m 2 -k n

The output file would be sim_SV_genome_1.fq and sim_SV_genome_2.fq