User Manual

simuG: a general-purpose genome simulator

Release v1.0.0
(2018-12-10)

Author Contact:

Jia-Xing Yue (岳家兴)

Email: yuejiaxing@gmail.com
GitHub: yjx1217
Twitter: iAmphioxus
Website: http://www.iamphioxus.org

Table of Contents

Introduction

Along with the rapid progressing of genome sequencing technologies, many bioinformatics tools have been developed for characterizing genomic variants based on genome sequencing data. While there is an increasing availability of experimentally validated gold-standard genome sequencing data set from real biological samples, *in silico* simulation remains a powerful approach for gauging and comparing the performance of bioinformatics tools. Here we developed a general-purpose genome simulator **simuG**, which is versatile enough to simulate both small (i.e. SNPs, INDELs) and large (i.e. CNVs, inversions, and translocations) genomic variants while staying light weighted with no extra dependency and minimal input requirements. These features together make simuG highly amenable to a wide range of application scenarios

simuG is a command-line tool written in Perl and supports all mainstream operating systems. It takes the user-supplied reference genome (in FASTA format) as the working template to introduce non-overlapping genomic variants of all major types (i.e. SNPs, INDELs, CNVs, inversions, and translocations). SNP and INDELs can be introduced in the same time, whereas CNVs (implemented as segmental duplications and deletions), inversions, and translocations can be introduced with independent runs. For each variant type, simuG can simulate pre-defined or random variants depending on specified options. For pre-defined variants, a user-supplied VCF file that specifies all desired variants is needed, based on which simuG will operate on the input reference genome to introduce the corresponding variants. For random variants, simuG provides a rich array of options for fine-grained controls. An ancillary script vcf2model.pl is further provided to directly calculate the best parameter combinations for the random SNP/INDEL simulation based on real data.

Citation

Jia-Xing Yue and Gianni Liti. (2018) simuG: a general-purpose genome simulator. (submitted; preprint available at https://www.biorxiv.org/content/early/2018/12/09/491498).

License

simuG is distributed under the MIT license.

Installation

simuG is implemented in Perl5 and does not have any extra dependency. So as long as Perl5 has been installed on your system, whether it is Linux, Mac OSX or Windows, you should be able to directly run simuG via the command-line interface on your system after downloading it from GitHub:

```
git clone https://github.com/yjx1217/simuG.git
cd simuG
perl simuG.pl -h
perl vcf2model.pl -h
```

Please note that GNU Gzip (https://www.gnu.org/software/gzip/) needs to be pre-installed in your system if you want to run simuG.pl and vcf2model.pl with compressed input files (*.gz).

What's Inside

Inside the downloaded simuG directory, you should see the following file structure:

```
.
├── LICENSE.md
├── README.md
├── Manual.pdf
├── simuG.pl
├── Testing_Example
│   ├── excluded_chr_list.yeast.txt
│   ├── sample.input.CNV.vcf.gz
│   ├── sample.input.INDEL.vcf.gz
│   ├── sample.input.inversion.vcf.gz
│   ├── sample.input.SNP.vcf.gz
│   ├── sample.input.translocation.vcf.gz
│   ├── SGDref.R64-2-1.centromere.gff3
│   ├── SGDref.R64-2-1.fa.gz
│   ├── sample.input.SNP.model.txt
│   ├── sample.input.INDEL.model.txt
│   └── Ty1_Ty3.breakpoint.gff3
└── vcf2model.pl
```

Two Perl scripts (*.pl) are provided, among which simuG.pl is the main program while vcf2model.pl is an ancillary script that can extract real-data based SNP and INDEL parameters from user-supplied SNP/INDEL variant calling VCF file. The directory Testing_Example contains some sample input files for walking through the testing examples.

Quick Start

Check the full list of available options of simuG.

```
perl simuG.pl -h
```

Simulate genome with pre-defined SNPs specified in the input VCF file.

```
perl simuG.pl \
     -refseq ./Testing_Example/SGDref.R64-2-1.fa(.gz) \
     -snp_vcf ./Testing_Example/sample.input.SNP.vcf(.gz) \
     -prefix output_prefix
```

Simulate genome with pre-defined INDELs specified in the input VCF file.

```
perl simuG.pl \
     -refseq ./Testing_Example/SGDref.R64-2-1.fa(.gz) \
     -indel_vcf ./Testing_Example/sample.input.INDEL.vcf(.gz) \
     -prefix output_prefix
```

Simulate genome with pre-defined CNVs specified in the input VCF file.

```
perl simuG.pl \
     -refseq ./Testing_Example/SGDref.R64-2-1.fa(.gz) \
     -cnv_vcf ./Testing_Example/sample.input.CNV.vcf(.gz) \
     -prefix output_prefix
```

Simulate genome with pre-defined inversions specified in the input VCF file.

```
perl simuG.pl \
     -refseq ./Testing_Example/SGDref.R64-2-1.fa(.gz) \
     -inversion_vcf ./Testing_Example/sample.input.inversion.vcf(.gz) \
     -prefix output_prefix
```

Simulate genome with pre-defined translocations specified in the input VCF file.

```
perl simuG.pl \
     -refseq ./Testing_Example/SGDref.R64-2-1.fa(.gz) \
     -translocation_vcf ./Testing_Example/sample.input.translocation.vcf(.gz) \
     -prefix output_prefix
```

Simulate genome with 1000 random SNPs.

```
perl simuG.pl \
     -refseq ./Testing_Example/SGDref.R64-2-1.fa(.gz) \
     -snp_count 1000 \
     -prefix output_prefix
```

Simulate genome with 100 random INDELs.

```
perl simuG.pl \
     -refseq ./Testing_Example/SGDref.R64-2-1.fa(.gz) \
     -indel_count 100 \
     -prefix output_prefix
```

Simulate genome with 10 random CNVs.

```
perl simuG.pl \
     -refseq ./Testing_Example/SGDref.R64-2-1.fa(.gz) \
     -cnv_count 10 \
     -prefix output_prefix
```

Simulate genome with 5 random inversions.

```
perl simuG.pl \
     -refseq ./Testing_Example/SGDref.R64-2-1.fa(.gz) \
     -inversion_count 5 \
     -prefix output_prefix
```

Simulate genome with 2 random translocations.

```
perl simuG.pl \
     -refseq ./Testing_Example/SGDref.R64-2-1.fa(.gz) \
     -translocation_count 2 \
     -prefix output_prefix
```

Additional Examples

Some more advance examples are provided as follows:

Check the full list of available options of the ancillary script vcf2model.pl.

```
perl vcf2model.pl -h
```

Use vcf2model.pl to generate SNP and INDEL models based on the input SNP/INDEL variant calling VCF file derived from real data.

```
perl vcf2model.pl \
     -vcf input.real_data.SNP_INDEL.vcf(.gz) \
     -prefix output_prefix
```

Use vcf2model.pl to generate SNP and INDEL models based on the input SNP/INDEL variant calling VCF file derived from real data while excluding variants called on the mitochondrial genome “chrMT” (defined in the excluded_chr_list.yeast.txt file) as well as variants with quality scores lower than 30 (if calculated).

```
perl vcf2model.pl \
     -vcf input.real_data.SNP_INDEL.vcf(.gz) \
     -qual 30 \
     -excluded_chr_list ./Testing_Example/excluded_chr_list.yeast.txt \
     -prefix output_prefix
```

Simulate genome with 1000 random SNPs and 100 random INDEL with titv_ratio = 2.0 and chrMT excluded (defined in the excluded_chr_list.yeast.txt file) for SNP simulation.

```
perl simuG.pl \
     -refseq ./Testing_Example/SGDref.R64-2-1.fa(.gz) \
     -snp_count 1000 \
     -titv_ratio 2.0 \
     -indel_count 100 \
     -excluded_chr_list ./Testing_Example/excluded_chr_list.yeast.txt \
     -prefix output_prefix
```

Simulate genome with 100 random CNVs with a biased copy number gain/loss ratio of 2.0 and chrMT excluded (defined in the excluded_chr_list.yeast.txt file). Also, centromeres of the reference genome has been specified, so the simulated CNVs will not disrupt these specified centromeres.

```
perl simuG.pl \
     -refseq ./Testing_Example/SGDref.R64-2-1.fa(.gz) \
     -cnv_count 100 \
     -cnv_gain_loss_ratio 2.0 \
     -excluded_chr_list ./Testing_Example/excluded_chr_list.yeast.txt \
     -centromere_gff ./Testing_Example/SGDref.R64-2-1.centromere.gff3 \
     -prefix output_prefix
```

Simulate genome with 5 random inversions using only specified genomic features (i.e. full-length Ty1 and Ty3 transposable elements in this example) as the potential breakpoints. The mitochondrial genome chrMT (defined in the excluded_chr_list.yeast.txt file) has been excluded for this simulation. Also, centromeres of the reference genome has been specified, so the simulated inversions will not disrupt these specified centromeres.

```
perl simuG.pl \
    -refseq ./Testing_Example/SGDref.R64-2-1.fa(.gz) \
    -inversion_count 5 \
    -inversion_breakpoint_gff ./Testing_Example/Ty1_Ty3.breakpoint.gff3 \
    -excluded_chr_list ./Testing_Example/excluded_chr_list.yeast.txt \
    -centromere_gff ./Testing_Example/SGDref.R64-2-1.centromere.gff3 \
    -prefix output_prefix
```

Simulate genome with 2 random translocations using only specified genomic features (i.e. full-length Ty1 and Ty3 transposable elements in this example) as the potential breakpoints. The mitochondrial genome chrMT (defined in the excluded_chr_list.yeast.txt file) has been excluded for this simulation. Also, centromeres of the reference genome has been specified, so the simulated translocations will not disrupt these specified centromeres.

```
perl simuG.pl \
    -refseq ./Testing_Example/SGDref.R64-2-1.fa(.gz) \
    -translocation_count 2 \
    -translocation_breakpoint_gff ./Testing_Example/Ty1_Ty3.breakpoint.gff3 \
    -excluded_chr_list ./Testing_Example/excluded_chr_list.yeast.txt \
    -centromere_gff ./Testing_Example/SGDref.R64-2-1.centromere.gff3 \
    -prefix output_prefix
```

Full Option List

```
Options:
    -help or -h
        Print help message.
        Example: -h.

    -man or -m
        Print more detailed help message.
        Example: -m.

    -version or -v
        Print version information.
        Example: -v.

    -refseq or -r
        Specify the reference genome to be used as the template (in fasta or
        fasta.gz format). This option is mandatory.
        Default = "".
        Example: -refseq ref.genome.fa(.gz).

    -snp_vcf
        Specify the list of exact SNP variants to be introduced (in vcf
        or vcf.gz format). When specified, the options '-snp_count',
        '-snp_model', and '-titv_ratio' will be ignored. If there are also
        INDEL variants in the vcf file, they will be automatically ignored.
        Default = "".
        Example: -snp_vcf snp.vcf(.gz).

    -snp_count
        Specify the number of SNP variants to be introduced.
        Default = "".
        Example: -snp_count 5000.

    -snp_model
        Specify the SNP model file generated by the ancillary script vcf2model.pl.
        When specified, the option '-titv_ratio' will be ignored.
        Default = "".
        Example: -snp_model snp_model.txt.

    -titv_ratio
        Specify the Ti/Tv ratio (transition/transversion ratio) used for
        simulate SNP variants.
        Default = 0.5.
        Example: -titv_ratio 2.0.

    -indel_vcf
        Specify the list of exact INDEL variants to be introduced (in vcf or vcf.gz
        format). When specified, the options '-indel_count', '-indel_model',
        '-ins_del_ratio', '-indel_size_powerlaw_alpha',
        and '-indel_size_powerlaw_constant' will be ignored.
        If there are also SNP variants in the vcf file, they will be automatically
        ignored.
        Default = "".
        Example: -indel_vcf indel.vcf(.gz).

    -indel_count
        Specify the number of INDEL variants to be introduced.
        Default = "".
        Example: -indel_count 500.
```

```
-indel_model
    Specify the INDEL model file generated by the ancillary script
    vcf2model.pl. When specified, the options '-ins_del_ratio',
    '-indel_size_powerlaw_alpha', and '-indel_size_powerlaw_constant'
    will be ignored.
    Default = "".
    Example: -indel_model indel_model.txt.

-ins_del_ratio
    Specify the Insertion/Deletion ratio used for simulate INDEL variants.
    Default = 1.0.
    Example: -ins_del_ratio 1.0.

-indel_size_powerlaw_alpha
    Specify the exponent factor alpha for power-law-fitted indel size
    distribution: p = C * (size) ** (alpha) for size >= 1 & size <= 50.
    Default = 2.0.
    Example: -indel_size_powerlaw_alpha 2.0.

-indel_size_powerlaw_constant
    Specify the exponent factor alpha for power-law-fitted indel size
    distribution: p = C * (size) ** (alpha) for size >= 1 & size <= 50.
    Default = 0.5.
    Example: -indel_size_powerlaw_constant 0.5.

-cnv_vcf
    Specify the list of exact CNV variants to be introduced (in vcf or
    vcf.gz format). When specified, the options '-cnv_count',
    '-cnv_gain_loss_ratio', '-cnv_max_copy_number', '-cnv_min_size',
    and '-cnv_max_size' will be ignored.
    Default = "".
    Example: -cnv_vcf cnv.vcf(.gz).

-cnv_count
    Specify the number of CNV variants to be introduced.
    Default = "".
    Example: -cnv_count 50.

-cnv_gain_loss_ratio
    Specify the relative ratio of DNA again over DNA loss.
    Default = 1.0.
    Example: -cnv_gain_loss_ratio 1.0.

-cnv_max_copy_number
    Specify the maximal copy number for CNV.
    Default = 10.
    Example: -cnv_max_copy_number 10.

-cnv_min_size
    Specify the minimal size (in basepair) for CNV variants.
    Default = 100.
    Example: -cnv_min_size 100.

-cnv_max_size
    Specify the maximal size (in basepair) for CNV variants.
    Default = 100000.
    Example: -cnv_max_size 100.
```

```
-inversion_vcf
    Specify the list of exact inversions to be introduced (in vcf
    or vcf.gz format). When specified, the options
    '-inversion_count', '-inversion_min_size',
    '-inversion_max_size', and '-inversion_breakpoint_gff' will be ignored.
    Default = "".
    Example: -inversion_vcf inversion.vcf(.gz).

-inversion_count
    Specify the number of inversions to be introduced.
    Default = "".
    Example: -inversion_count 5.

-inversion_min_size
     Specify the minimal size (in basepair) for inversion.
     Default = 1000.
     Example: -inversion_min_size 1000.

-inversion_max_size
     Specify the maximal size (in basepair) for inversion.
     Default = 1000000.
     Example: -inversion_max_size 1000000.

-inversion_breakpoint_gff
     Specify the list of potential breakpoints for triggering
     inversions (in gff3 or gff3.gz format).
     Default = "".
     Example: -inversion_breakpoint_gff inversion_breakpoint.gff(.gz).

-translocation_vcf
      Specify the list of exact translocations to be introduced (in vcf
      or vcf.gz format). When specified, the options '-translocation_count'
      and '-translocation_breakpoint_gff' will be  ignored.
       Default = "".
      Example: -translocation_vcf transloaction.vcf(.gz).

-translocation_count
       Specify the number of translocations to be introduced.
       Default = "".
       Example: -translocation_count 1.

-translocation_breakpoint_gff
    Specify the list of potential breakpoints for triggering
    translocations (in gff3 or gff3.gz format).
    Default = "".
    Example: -translocation_breakpoint_gff translocation_breakpoint.gff(.gz).

-centromere_gff
    Specify centromeres for constraining the CNV, inversion, and
    translocation simulation (in gff3 or gff3.gz format).
    Default = "".
    Example: -centromere_gff centromere.gff(.gz).

-excluded_chr_list
    Specify the name of chromosome(s) to be excluded for introducing
    genomic variants (a single-column list file in txt format).
    Default = "".
    Example: -excluded_chr_list excluded_chr_list.txt.
```

```
-seed or -s
    Specify an integer as the random seed for the simulation.
    Default = "".
     Example: -seed 201812.

-prefix or -p
    Specify the prefix for output files.
    Default = "output_prefix".
    Example: -prefix sim.
```

Inputs

For both pre-defined and random variant simulation, an input reference genome in FASTA format is mandatory. As an example, we provided a sample reference genome file (SGDref.R64-2-1.fa.gz) in the Testing_Example directory, which is the reference genome of the budding yeast *Saccharomyces cerevisiae* S288C (vR64-2-1). simuG will also evaluate the base composition of the input reference genome and use it for derive insertion sequence for random INDEL simulation .

For pre-defined variant simulation, a VCF file is needed to specify the genomic variants that you want to introduce in the simulation. Please see those sample.input.*.vcf.gz files in the Testing_Example directory as examples. For SNPs and INDELs, this should be vary straightforward. Only the information from the 'CHROM', 'POS', 'REF', and 'ALT' will be used for SNP and INDEL simulation. Only homozygous SNPs and INDELs will be used. For structural variants such as CNVs, inversions, and translocations, special attention needs to be paid given the complexity of denoting different types of SVs using the VCF file format. For general guidance, please check VCF's official specification (http://samtools.github.io/hts-specs/VCFv4.1.pdf). For simuG, we use the short-hand notation to denote segmental deletions (introduced during CNV simulation) and inversions. Information from the 'CHROM' and 'POS' fields as well as the 'SVTYPE=', 'EVENT=' and 'END=' tags from the 'INFO' filed will be used. The VCF breakend notation is used to represent segmental duplications (introduced during CNV simulation) and translocations due to the involvement of multiple genomic regions. Therefore, there will be two lines of records to represent each segmental duplication (recording the new integration location) and there will be four lines of records to represent each translocation. Information from the 'CHROM', 'POS', and 'ALT' fields as well as the 'SVTYPE=', 'EVENT=' tags will be used. For the 'ALT' field, only the "chr:pos" information will be used by simuG. Finally, we want to stress that for the CNV, inversion, and translocation simulation, the value of the 'EVENT=' tag should always be unique for each event. When preparing your own input VCF files for simuG's pre-defined CNV, inversion, and translocation simulation, please take the sample VCFs that we provided in the Testing_Example directory as the example.

For random variant simulation, you can specify the genomic location of centromeres as well as the preferred inversion and translocation breakpoints using GFF3 files to constrain the CNV, inversion, and translocation simulation. A specification for the GFF3 format can be found here: http://gmod.org/wiki/GFF3#GFF3_Format. For centromeres, simuG can extract the genomic locations of centromeres from the specified centromere GFF3 file so that simulated random CNVs, inversions, and translocations will not disrupt these centromeres. For breakpoints, the value of both feature type (column 3) and feature strand (column 7) will be considered when sampling breakpoint pairs for inversions or translocations. For example, the sampled breakpoint pairs that can trigger inversion should belong to the same repetitive sequence type (column 3) but from opposite strands (column 7). Also, when specified, centromere will be given special consideration in random translocation simulation so that translocations leading to dicentric chromosomes will not be sampled. In the Testing_Example directory, we provided examples of both centromere GFF3 file (SGDref.R64-2-1.centromere.gff3) and breakpoint GFF3 file (Ty1_Ty3.breakpoint.gff3) for your check. By using this sample breakpoint GFF3 file, we constraint simuG to only use specified full-length Ty1 and Ty3 transposable elements as potential inversion/translocation breakpoints. As you can see here, you can also define breakpoints from different feature types (reflected by column 3) in the same file. In this case, simuG will further classify these potential breakpoints into different subgroups based on their respective feature types (i.e. Ty1 and Ty3) and only sampling breakpoints within each subgroup.

In addition, when needed, users can provide simuG a single-column list file for specifying a list of chromosome(s) to be excluded from variant introduction. We provided such an example (excluded_chr_list.yeast.txt) in the Testing_Example directory. In this example, we exclude the mitochondrial genome chrMT in our simulation. Finally, the random SNP and INDEL simulation can take the pre-computed SNP and INDEL model files (e.g. sample.input.SNP_model.txt and sample.input.INDEL_model.txt in the Testing_Example directory) produced by the ancillary script vcf2model.pl.

Outputs

Upon the completion of the simulation, three files will be produced: 1) a simulated genome bearing introduced variants in FASTA format, 2) a tabular file showing the genomic locations of all introduced variants relative to both reference genome and simulated genome, 3) a VCF file showing the genomic locations of all introduced variants relative to the reference genome.

!!! IMPORTANT Please note that in order to keep records on the exact genomic locations of introduced variants, simuG does not perform variant normalization for the generated VCF files. Depending on the immediate neighboring bases of the introduced genomic variants, this might have an impact if you directly compare simuG's VCF outputs with those from other tools. Therefore, it is highly recommended to first normalize simuG's VCF outputs as well as the VCF outputs from other tools using a VCF normalization tool such as vt (https://github.com/atks/vt) before making such comparison. You can read more about variant normalization here (https://genome.sph.umich.edu/wiki/Variant_Normalization).

Visualization

For visualizing simulated gross chromosomal rearrangements such as inversions and translocations, we recommend using D-Genies (http://dgenies.toulouse.inra.fr/) to produce interactive dotplots between your input reference genome and simuG's simulated genome. D-Genies comes with a native online web interface (http://dgenies.toulouse.inra.fr/run) to handle user-submitted jobs, which is very convenient for the end users. Alternatively, you can also install D-Genies locally for batch job processing. Alternative tools such as Mummer (https://github.com/mummer4) and Gepard (http://cube.univie.ac.at/gepard) can also make nice dotplots for such pairwise genome comparison, although a little bit more work will be needed.

Tips

1) With the exception of SNP and INDELs which can be simulated simultaneously, simuG simulates one type of genomic variation a time to avoid the complication of tracking different types of variants in the same time. In real case scenario, sometimes you might want to introduce multiple types of variants into the simulated genome. This can be easily achieved by running simuG multiple times with a sequential fashion. So use the output of the previous run as the input of the next run.

2) For simulating pre-defined CNVs, inversions, and translocations, it is often the case that you know the coordinate of the genomic region for the rearrangement but do not know the exact nucleotide bases at the breakpoint location. The latter is needed in standard VCF notations. Since simuG can directly extract this information based on the specified genomic coordinate, you can just use '.' to represent the corresponding nucleotide base when preparing your own input VCF files. For example, the standard VCF breakend notation:

```
chrX  382742  .  T  T]chrXII:961373]  .  .  SVTYPE=BND;EVENT=TRA_1
chrX  382743  .  A  [chrXII:961374[A  .  .  SVTYPE=BND;EVENT=TRA_1
```

is equivalent to the following simplified version for simuG:

```
chrX  382742  .  .  .]chrXII:961373]  .  .  SVTYPE=BND;EVENT=TRA_1
chrX  382743  .  .  [chrXII:961374[.  .  .  SVTYPE=BND;EVENT=TRA_1
```