Genomator

Genomator is a program which uses a SAT solver to generate synthetic genomic data from an input VCF dataset. The program parses an input file of genomic data for a number of individuals, specified in VCF format, and then generates the genomic data for a number of synthetic individuals, outputting the results as a different VCF formatted file. Genomator formulates SAT problem (boolean SATisfiability problem) which is composed of a (usually) large number of binary variables and a (usually) large number of constraints. In this context the SAT problem is representative of generating new individuals which are sufficiently similar to those in the dataset and yet also sufficiently different from each.

In the following sections, we give information on the **1. Installation** and **2. Invocation** of the Genomator program, outlining its **3. Configuration** and the **4. Mechanism** of its operation.

1. Installation

Genomator is written in the Python programming language, and requires an installation of a relatively recent version of Python3. You can download and install Python from directions of the python homepage (https://www.python.org/). Additionally Genomator requires a number of python libraries to be installed, it is advisable to install the libraries which a program such as Genomator uses inside a *Virtual Environment*. A Virtual Environment is a local directory which stores relevant libraries separately from those installed on the wider system. The process of creating and using a Virtual Environment from a command console is detailed in the python user guide (https://packaging.python.org/en/latest/guides/installing-using-pip-and-virtual-environments/). Once you have created and activated a Virtual Environment (or otherwise), you should have the **pip** command available from the commandline, which will allow you to install relevant libraries for using Genomator.

Genomator requires the following libraries (their approximate versions and tacitly also their dependencies):

1. tqdm ~= 4.64.1
2. vcfpy ~= 0.13.6
3. python-sat ~= 0.1.7
4. click ~= 8.1.3
5. cyvcf2 ~= 0.30.18
6. stopit ~= 1.1.2

You can install Genomator itself and its libraries by typing the following command from the appropriate directory (including the necessary "setup.py" file)

```
pip install .
```

This command should install Genomator and its associated libraries.

2. Invocation

Once the software has been installed, Genomator can be invoked. As may be demonstrated by loading its help page:

```
python genomator.py --help
```

This command should display a help message and program description, an actual invocation of the Genomator in its capacity to generate genomes can be invoked by executing the *run.sh* script from the commandline

```
./run.sh
```

Which should trigger a basic invocation as:

```
python genomator.py input-small.vcf my_vcf_output.vcf 10 1 1
```

This command should quickly generate a VCF output file "my_vcf_output.vcf" of 10 individuals, from the input VCF file "input-small.vcf". The individuals output from this command invocation will be atleast marginally different from each of the input individuals, and marginally different from each other (by atleast 1 nucleotides in all cases). In this case, each new individual sample must be at least distance 1 (from the first 1) from each individual in the original data source, and atleast distance 1 (from the second 1) from those individual samples already generated (the synthetic ones).

Genomator has a range of options and configurations as detailed in the next section:

CUSTOM SAT SOLVER INSTALL - TINICARD Although optional, Genomator is connectable to a custom SAT solver tinicard, this option is set using the commandline flag:

```
--solver_name=tinicard
```

To enable this custom SAT solver, it needs to be installed. on a POSIX compatible system, with a recent GCC/G++ compatible compiler, and with Python Headers installed in the system, you enter the tinicard repository, and install with **pip**

```
cd tinicard
pip install .
```

CUSTOM SAT SOLVER INSTALL - CMSGEN Although optional, Genomator is also compatible with uniform-like sampler CMSGen, and is configurable using the commandline flag:

```
--solver_name=cmsgen
```

To enable this custom SAT solver, it needs to be installed.

```
pip install pycmsgen
```

3. Configuration

Genomator is invoked from the command line, where there is two mandatory arguments then three arguments with default values (and hence do not strictly need to be specified), and three additional specifiable options. Executing Genomator without these gives mandatory arguments gives you a summary of the required invocation format:

```
python genomator.py
```

Gives:

```
Usage: genomator.py [OPTIONS] INPUT_VCF_FILE OUTPUT_VCF_FILE
                    [NUMBER_OF_GENOMES] [DIVERSITY_REQUIREMENT]
                    [GENERATED_DIVERSITY_REQUIREMENT]
```

running

```
python genomator.py --help
```

Gives internal documentation of arguments/options as:

```
Usage: genomator [OPTIONS] INPUT_VCF_FILE OUTPUT_VCF_FILE
[NUMBER_OF_GENOMES]
                 [DIVERSITY_REQUIREMENT] [GENERATED_DIVERSITY_REQUIREMENT]

  /__ _ __  _ __  _ _|_ _  __
  \_|(/_| |(_)|||(_| |_(_) |

  GENOMATOR (pre-alpha 0.1.5)

  Authors: Mark A Burgess

  generates <NUMBER_OF_GENOMES> synthetic genomes from a dataset of an
  <INPUT_VCF_FILE> to an <OUTPUT_VCF_FILE> with <DIVERSITY_REQUIREMENT>
from
  the dataset. The generated genomes will each be different from each
other by
  <GENERATED_DIVERSITY_REQUIREMENT> for fun and profit!

  the <DIVERSITY_REQUIREMENT> and <GENERATED_DIVERSITY_REQUIREMENT> are
  integers that specifies that the resulting synthetic genomes must be
  different from supplied genomes/themselves by a minimum distance
```

```
(Hamming
  distance in logical space).

  optionally <REFORMATED_INPUT_VCF_FILE> will be the successfully parsed
lines
  of the <INPUT_VCF_FILE> optionally the <EXCEPTION_SPACE> (default 0) is
the
  additional strength of the discriminator (informing how similar the
  resulting genomes will be to dataset). optionally specifying the
  <SOLVER_NAME> to use a different SAT solver from the PYSAT python
library.
  (note: not all solvers support cardinality constraints nessisary for
  operation)

  NOTE: not all genomes/diversity_requirements are possible (owing to
current
  query structure (can be weakened in future) you need ~> 16 genomes to be
  able to meaningfully diversify on random data)

Options:
  --solver_name
[cd|cd103|cdl|cdl103|cadical103|cd15|cd153|cdl15|cdl153|cadical153|cd19|cd
195|cdl19|cdl195|cadical195|cms|cms5|crypto|crypto5|cryptominisat|cryptomi
nisat5|g3|g30|glucose3|glucose30|g4|g41|glucose4|glucose41|g42|g421|glucos
e42|glucose421|gc3|gc30|gluecard3|gluecard30|gc4|gc41|gluecard4|gluecard41
|lgl|lingeling|mcb|chrono|chronobt|maplechrono|mcm|maplecm|mpl|maple|maple
sat|mg3|mgs3|mergesat3|mergesat30|mc|mcard|minicard|m22|msat22|minisat22|m
gh|msat-gh|minisat-gh|tinicard|cmsgen]
  --exception_space FLOAT
  --indexation_bits INTEGER
  --sample_group_size INTEGER
  -nc, --no_smart_clustering      disables smart clustering
  -di, --del_info_field           Disables copying the INFO field on
output
  -nb, --no_biasing               Disables phase biasing for tinisat
  --cluster_information_file PATH
  --difference_samples INTEGER
  --looseness FLOAT
  --max_restarts INTEGER
  -da, --dump_all_generated       dump generated individuals if
max_restarts
                                  is hit
  --involutions INTEGER
  --tasks INTEGER
  --noise FLOAT
  --kull INTEGER
  --help                          Show this message and exit.
```

Which should be relatively self explanatory, but going into detail on each option: Mandatory arguments are:

- **INPUT_VCF_FILE:** is the path to a valid *.vcf* file of genetic data for the individuals constituting our dataset (you can read more about *vcf* format: https://samtools.github.io/hts-specs/VCFv4.2.pdf, https://en.wikipedia.org/wiki/Variant_Call_Format)
- **OUTPUT_VCF_FILE:** is a path to a *.vcf* file that will be created (and/or overwritten) by the genomator program, dumping a similarly formatted *.vcf* file with information of the individuals generated, the individuals will be annotated by sample_name "GENERATEDSAMPLE{X}" where X is sequential number.

Optional Arguments are:

- **NUMBER_OF_GENOMES:** is an integer value (default 1) of the number of individuals the Genomator program will attempt to generate, eg. if you wish Genomator to generate 15 individuals, this number will be 15. (**NOTE:** if this number is "0" or input as negative, then Genomator will enter into a mode of attempting to generate as many individuals as it can, until it is interrupted by an exception or "CTRL-C" cancel command is given)
- **DIVERSITY_REQUIREMENT:** the required minimum distance (default 0) between generated individuals and each of the individuals in the dataset, the distance is the Hamming distance, in logical space (see mechanism section for precise definition) - decreasing this distance means that the Genomator solver can create sequences that are closer to what is in the dataset, and the solution process is less constrained and easier, but increasing it means that the generated individuals are constrained to be generated more differently than the dataset, but makes the solution process harder (or even impossible, if it is too high)
- **GENERATED_DIVERSITY_REQUIREMENT:** is the required minimum distance (default 0) between generated individuals between themself, this distance is the Hamming distance in logical space (see mechanism section for precise definition) - decreasing this distance means that the individuals which Genomator generates can be closer together (or even identical if this distance is zero), while increasing this distance requires that those individuals will be more different from each other, the higher this number is the harder the problem is (or even impossible, if it is too high), Genomator will iteratively generated individuals upto NUMBER_OF_GENOMES specified, with each additional genome generated, this GENERATED_DIVERSITY_REQUIREMENT will further constrain the problem of generating a new individual - which may slow the problem down (or make it impossible)

Additionally there is a couple of options available:

- **solver_name:** specified as "--solver_name=[...]" this gives the option of swapping out the underlying SAT solver that Genomator uses, the respective SAT solver should be able to handle cardinality constraints (and will error if it cannot), swapping out the solver may result in faster processing times, but may also influence spread of the resulting individuals generated (as these solvers may handle randomisation differently), currently Minisat (default) is used, but any of the glue-card solvers should work as well.
- **exception_space:** specified as "--exception_space=X", the exception space parameter is optional integer (defaulting to benign 0), is specifiable to add extra strength to the indistinguishability constraint (see mechanism section for further information), setting this to non-zero (or higher) will result in a more constrained problem (and will take longer to generate the problem prior to solving), and will make the individuals that Genomator generates more similar to those in the dataset - while keeping the indistinguishability constraints to 2nd order.
- **indexation_bits:** only effective with the Tinicard SAT solver, controls how many bits of information are used as an index in the dynamic creation of SAT clauses

- **sample_group_size:** the size of each cluster of inputs taken into Genomator algorithm for each output generated.
- **no_smart_clustering:** by default the clustering method is based on Hamming distance between genomes, however enabling this flag makes the clusters generated by random sampling
- **del_info_field:** deletes the INFO field in the output VCF file
- **no_biasing:** only effective with the Tinicard SAT solver, disables the deliberate biasing to recreate SNP frequencies of the input data
- **cluster_information_file:** the path to a file for-which the information about generated clusters which are used can be dumped to
- **difference_samples:** In the process of clustering genomes for input into the Genomator algorithm, the number of SNPs to consider in approximation of Hamming distances
- **looseness:** a floating point number and the probability that a pair of SNP features in the input will propagate as a constraint to affect the generation of the output (essentially a noise introduction feature)
- **max_restarts:** the number of time Genomator will restart itself before failing if it cannot generate a genome (often because constraints are too tight)
- **dump_all_generated:** If maximum restarts do occur, and Genomator fails to generate the required NUMBER_OF_GENOMES, should it dump whatever genomes it has generated?
- **involutions:** how many times should Genomator generate new synthetic genomes from the synthetic genomes it generates.
- **tasks:** how many compute tasks should Genomator attempt to employ in parallel
- **noise:** what proportion of white noise should Genomator add to its output
- **kull:** if specified, how many input samples should be considered at all.
- **help:** specified as "--help" without a parameter, it displays the help message