EnivolCrossing Manual

Perry Xu, Shenni Liang, Jae Hahn, Vivian Zhao

Update on February 21, 2024

Contents

Glossary 5

Preface 7
- 0.1 An Overview of EnivolCrossing 7
  - 0.1.1 Introduction 7
  - 0.1.2 A quick summary of usage 7
- 0.2 Installation 8
  - 0.2.1 Docker 8
  - 0.2.2 conda 8
- 0.3 Copyright and License 8

I Pre-simulation programs 9

1 Overview 11

2 Run each program separately 13
- 2.1 NetworkGenerator 13
  - 2.1.1 User input 14
  - 2.1.2 Uniform host population 15
  - 2.1.3 Superspreaders 15
  - 2.1.4 Two Sub-populations 16
- 2.2 SeedGenerator 16
  - 2.2.1 User input 19
  - 2.2.2 burn-in by Wright-Fisher model 19
  - 2.2.3 burn-in by epidemiological model 20
- 2.3 GeneticEffectGenerator 21
  - 2.3.1 User input 23
  - 2.3.2 Generate from GFF file 23
- 2.4 HostSeedMatcher 24
  - 2.4.1 User-provided matching file input 25
  - 2.4.2 Generate matching based on user-specified methods & parameters 26

II Simulator 27

3 Overview of the usage 29
- 3.1 Run OutbreakSimulator 29
- 3.2 Configuration file 30
  - 3.2.1 BasicRunConfiguration 30
  - 3.2.2 EvolutionModel 30
  - 3.2.3 SeedsConfiguration 31
  - 3.2.4 GenomeElement 31
  - 3.2.5 NetworkModelParameters 31
  - 3.2.6 EpidemiologyModel 31

4 Modules of the simulator 35
- 4.1 Evolutionary model 35
- 4.2 Genetic effects to fitness 35
- 4.3 Compartmental model 36
- 4.4 Epoch structure 37
- 4.5 Massive sampling events 37

5 Specifying your own configuration 39
- 5.1 A minimal model for transmissibility 39
- 5.2 Multi-stage drug treatment 42

III Process the output data 43

6 Output structures 45
- 6.1 Reading the output plots 45

IV Streamline 47

V GUI 49

VI Examples 51

Glossary

drug-resistance	The probability of treatment failure on each pathogen induced by its genome.. 21, 30, 35
epoch	All ticks in the OutbreakSimulator can be divided into more than one epochs, all rates parameters and the effect size traits can be set differently for each epoch.. 22, 23, 31, 32, 37
host	The larger organism that harbours one or more pathogens. 7, 31
pathogen	The organism whose movement and genomes XXX is modeling. 7
seed	The pathogen genome(s) that infect a fully susceptible host population at first tick of XXX's outbreak simulation.. 16, 31
super-infection	The circumstance new host(s) infected the other host when the infected hosts hasn't recovered from a previous infection, the multiple infections could happen at the same tick or different ticks.. 30
tick	Taken from the tick defined in SLiM. It is the time unit of all the epidemiological events, and the time unit of mutation, and the branch length unit of the transmission tree.. 17, 30, 32
transmissibility	The relative transmission probability of each pathogen induced by its genome.. 21, 30, 32, 35

Preface

Contents

0.1 An Overview of EnivolCrossing . . . 7
0.2 Installation . . . 8
0.3 Copyright and License . . . 8

0.1 An Overview of EnivolCrossing

EnivolCrossing (**E**p**I**demiological a**N**d e**VOL**utionary process **C**oupl**ING** **S**imulato**R**) is an outbreak simulator that provides a flexible approach to model pathogen evolution under different epidemiological scenarios, featuring the coupling of evolutionary process of pathogen genomes and the outbreak, with demography components. EnivolCrossing is composed of different pre-simulation modules that generates host population structure, pathogen genome sequences and fitness calculation by different models.

0.1.1 Introduction

EnivolCrossing implements an agent-based, discrete and forward in time approach to simultaneously simulating the transmission dynamics and molecular evolution of haploid transmissible pathogen population within a static host population. EnivolCrossing highlights the coupling of evolutionary process and epidemiological process and explicitly models transmission on top of a host contact network. Utilizing a compartmental model framework, EnivolCrossing adeptly balances the simplification of real-world processes with the theoretical constructs inherent in both epidemiological and quantitative genetic models.

The EnivolCrossing software integrates a combination of Python, Bash, R, and SLiM scripts. The first part of the software is designed for generating all essential input files for initiating a simulation run in EnivolCrossing. The second part executes the main simulation process using SLiM, a script-based simulator that is designed for population genetics at back-end. By generating appropriate SLiM scripts per configuration, different epidemiological scenarios could be simulated, including different permissible transitions between compartments and different treatment stages affected by genetic-based drug resistance. The third part of the software includes analysis and visualizations for the simulation output, providing with users raw data, summary statistics and graphic representations of results such as the phylogeny of transmission tree and the compartments' trajectories averaged over all simulation runs.

0.1.2 A quick summary of usage

EnivolCrossing provide three ways of execution per user need. There are 5 programs in EnivolCrossing (**??**): *NetworkGenerator* (Section 2.1), *SeedGenerator* (Section 2.2), *GeneticEffectGenerator* (Section 2.3), *HostSeedMatcher* (Section 2.4), *OutbreakSimulator* (Chapter 3).

- **Sequential**: Run each program individually and sequentially by providing parameters for each program.
- **Streamlined**: Run each program in a streamline by specifying a complete configuration file.
- **By GUI**: GUI is a user-friendly way of executing the programs interactively.

0.2 Installation

To check for updates and open issues, please refer to our github page EnivolCrossing by Kim Lab

0.2.1 Docker

```
git clone EnivolCrossing.git
## Build the dockerfile
docker build -t cmonjeau/slim .
## Run test
docker run -it --rm PATH EnivolCrossing -config test.config
```

Listing 1: Docker usage

0.2.2 conda

To install EnivolCrossing by conda, please .... You might need administrator right to successfully install the whole software.

```
conda install EnivolCrossing
##### Python version ?
### To test, run
EnivolCrossing -config test.config
```

Listing 2: Conda installation

0.3 Copyright and License

- GitHub Repo: https://github.com/EpiEvoSoftware/original_pipeline
- Cite: *A url of paper*
- Lab webpage: https://jaeheekimlab.github.io

Part I

Pre-simulation programs

Chapter 1

Overview

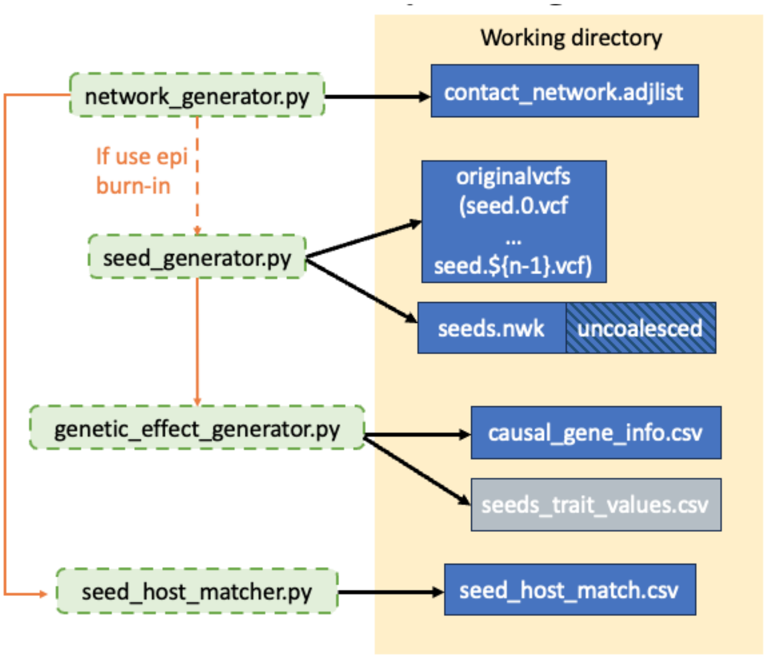
The *OutbreakSimulator* of EnivolCrossing needs the specification of some input files by running the pre-simulation programs sequentially. Here are the prerequesites needed before running all the programs:

- A **reference genome** in `fasta` format
- A **working directory** that stays consistent for running all the programs (All programs of EnivolCrossing requires an input of path to the working directory `-wkdir` in command line except for *OutbreakSimulator*, for which path to the working directory should be specified in the configuration file).

Other requirements are based on specific user need when running different modes of the programs. For example, if user want to let EnivolCrossing randomly generate a genetic architecture, then a `.gff`-like file will be needed when running *GeneticEffectGeneration* program. All the modes of each program and the required inputs for each mode will be introduced below.

Some modes of some programs depend on other programs to have been run, as they take input files from the specified working directory, and would throw an error when those files are not found or in an incorrect format. Here is a typical workflow of running all the pre-simulation programs:

1. Run *NetworkGenerator*. Output of this program might be the dependency files for *SeedGenerator* in some modes.
2. Run *SeedGenerator* if seeds with genomes different from the reference genome is required in your simulation. If you want to plant seeds with identical genomes as the reference genome, then this step can be omitted. Please refer to the simulator chapter about how to specify this model.
3. Run *GeneticEffectGenerator* if non-neutral mutations are used in your simulation. This step can be omitted with a non-genetic model, where genomes and mutations are still modeled but not affecting the fitness of the pathogens.
4. Run *SeedHostMatcher* after *NetworkGenerator* and *SeedGenerator* have already been run to match each seed to a specific hoat.


Working directory
network_generator.py
contact_network.adjlist
If use epi burn-in
seed_generator.py
originalvcfs
(seed.0.vcf
...
seed.${n-1}.vcf)
seeds.nwk
uncoalesced
genetic_effect_generator.py
causal_gene_info.csv
seeds_trait_values.csv
seed_host_matcher.py
seed_host_match.csv

Chapter 2

Run each program separately

Contents

2.1 NetworkGenerator 13
2.2 SeedGenerator 16
2.3 GeneticEffectGenerator 21
2.4 HostSeedMatcher 24

2.1 NetworkGenerator

EnivolCrossing models the host population and their contact network explicitly. *NetworkGenerator* is used to generate an adjacent list file in the working directory that stores the contact network information of hosts. To run *NetworkGenerator*, the required inputs are working directory (`-wkdir`), host population size (`-popsize`) and method of network input (`-method`). This is required to run either user wants to provide their own contact network, or user want to generate a random contact network. This is how to run *NetworkGenerator*:

```
python network_generator.py \
    -wkdir ${YOUR_WORK_DIR} \
    -popsize ${HOST_SIZE} \
    -method ${METHOD} \
    [Other params]
```

Listing 2.1: NetworkGenerator Usage

We are now introducing all the options:

- `wkdir`: [*REQUIRED] (`string`)
  Absolute path to the working directory, which should typically stays consistent for each program being run.

- `popsize`: [*REQUIRED] (`integer`)
  Size of the host population you want to model, i.e. Number of nodes in the random graph.

- `method`: [*REQUIRED] (`string`)
  The method of generating the network, which can only be one of the two methods: `user_input` or `randomly_generate`. By specifying `user_input`, you are required to provide your own contact network file by `-path_network`. By specifying `randomly_generate`, you are required to provide a random network model by `-model` and corresponding parameters.

- `path_network`: [*OPTIONAL] (`string`)
  Required if `method` is specified as `user_input`. Full path to the contact network file user provided. The file has to be in an adjacency list format.
- `model`: [*OPTIONAL] (`string`)
  Required if `method` is specified as `randomly_generate`. The random graph model you want to use for generating the contact network, which can only be one of the three: `ER` (Erdos-Renyi graph), `BA` (Barabarsi-Albert graph) or `RP` (Random partition graph).
- `p_ER`: [*OPTIONAL] (`float`)
  Required if `model` is specifed as `ER`. It is the parameter for generating an Erdos-Renyi graph, specifying the probability of presence of every possible edge.
- `rp_size`: [*OPTIONAL] (`list of int`)
  Required if `model` is specifed as `RP`. It is the parameter for generating a random partition graph, specifying the size of each partition. We only support a two-partition model, thus this option has to be an integer list of size 2, which sums up to be the size of the host population.
- `p_within`: [*OPTIONAL] (`list of floats`)
  Required if `model` is specifed as `RP`. It is the parameter for generating a random partition graph, specifying the probability of presence of every possible edge between nodes within the same partition. We only support a two-partition model, thus this option has to be a float list of size 2, each representing a probability for each partition.
- `p_between`: [*OPTIONAL] (`float`)
  Required if `model` is specifed as `RP`. It is the parameter for generating a random partition graph, specifying the probability of presence of every possible edge between nodes from different partitions.
- `m`: [*OPTIONAL] (`float`)
  Required if `model` is specifed as `BA`. It is the parameter for generating a Barabasi-Albert graph.

We are introducing logistics and examples of generating different host contact network in the following sections.

2.1.1 User input

If user want to provide their own contact network, user need to make sure they have a contact network file in an adjacency list format, which is a space-delimited file and lines starting with # will be marked as comments. By running this mode, a typical command is:

```
python network_generator.py \
    -wkdir ${YOUR_WORK_DIR} \
    -popsize ${HOST_SIZE} \
    -method user_input \
    -path_network ${YOUR_NETWORK_PATH}
```

Listing 2.2: NetworkGenerator user_input mode

NetworkGenerator will read in the network from the file path user provided and check its format, if it satisfies the requirements, the network will be rewritten into a file named `contact_network.adjlist` in the working directory. Note that *OutbreakSimulator* can't be run without a properly formatted and named `contact_network.adjlist` in the working directory. So this step is recommended even if you're providing your own contact network. However, you can modify `contact_network.adjlist` by adding/deleting new edges manually as well, but the *NetworkGenerator* is recommended to be rerun by taking the full path of `${YOUR_WORK_DIR}/contact_network.adjlist` as input for `-path_network` after your manual modification to make sure it's in a proper format except for non-canonical usage.

Non-canonical usage includes weighted graph, by default the contact network is unweighted, but you can add weight to an edge by duplicating the node ids on lines you want. Do not rerun the program or the weighted won't be pretained, but please make sure you are retaining the proper format when doing this.

2.1.2 Uniform host population

Often people want to generate a uniform contact network for a specified host population size, featuring a host population that is well-mixed, and every host has a same number of expected contacts, which can be calculated by $p_{ER} \times$ host size, which often serves as a null model for population structure. By running this mode, a typical command is:

```
python network_generator.py \
    -wkdir ${YOUR_WORK_DIR} \
    -popsize ${HOST_SIZE} \
    -method randomly_generate \
    -model ER \
    -p_ER ${PROB_ER}
```

Listing 2.3: NetworkGenerator ER network

NetworkGenerator will use the parameters provided to generate an Erdos-Renyi graph and write down the network into a file named `contact_network.adjlist` in the working directory. For examples, to generate a uniform host population of 10000 hosts and each host has an expected connection of 10 other hosts, we run:

```
python network_generator.py \
    -wkdir ${YOUR_WORK_DIR} \
    -popsize 10000 \
    -method randomly_generate \
    -model ER \
    -p_ER 0.001
```

Listing 2.4: NetworkGenerator ER network example

2.1.3 Superspreaders

Using this option to generate a host contact network that features an unbalanced connectivity, where only a few potential ”superspreaders” has a big connectivity and the rest having low connectivity, representative of a real-world social network. By running this mode, a typical command is:

```
python network_generator.py \
    -wkdir ${YOUR_WORK_DIR} \
    -popsize ${HOST_SIZE} \
    -method randomly_generate \
    -model BA \
    -m ${m}
```

Listing 2.5: NetworkGenerator BA network

NetworkGenerator will use the parameters provided to generate an Erdos-Renyi graph and write down the network into a file named `contact_network.adjlist` in the working directory. For examples, to generate a uniform host population of 10000 hosts and has $m = 2$, we run:

```
python network_generator.py \
    -wkdir ${YOUR_WORK_DIR} \
    -popsize 10000 \
    -method randomly_generate \
    -model BA \
    -m 2
```

Listing 2.6: NetworkGenerator BA network example

2.1.4 Two Sub-populations

Using this option to generate a host contact network for a population that has sub-population structures, where there are two partitions in the whole graph, each partitions has their own connecting probability within partition, and a probability of connecting between partitions, representative of a structures like a rural and urban area, or two countires. By running this mode, a typical command is:

```
python network_generator.py \
    -wkdir ${YOUR_WORK_DIR} \
    -popsize ${HOST_SIZE} \
    -method randomly_generate \
    -model RP \
    -p_within ${PROB_POP1} ${PROB_POP2} \
    -p_between ${PROB_BETWEEN_1_2}
```

Listing 2.7: NetworkGenerator RP network

NetworkGenerator will use the parameters provided to generate a random partition graph and write down the network into a file named `contact_network.adjlist` in the working directory. For examples, to generate a uniform host population of 10000 hosts, of which 6000 hosts come from a urban area where expected connected hosts for each host is high as 12, and the rest 4000 come from a rural area where expected connectivity for each host is as low as 4, and there are few connections between the two area so that each rural host has an expected urban host connection of 3, we run:

```
python network_generator.py \
    -wkdir ${YOUR_WORK_DIR} \
    -popsize 10000 \
    -method randomly_generate \
    -p_within 0.002 0.001 \
    -p_between 0.0005
```

Listing 2.8: NetworkGenerator RP network example

2.2 SeedGenerator

Seeds are pathogen genomes that are planted to the host population at the start of the simulation, which have their own genomic sequences and could have different fitness in terms of transmissibility and drug-resistance due to specific mutations they harbour. If user wish the simulation to start from the reference genomes and don't wish to have a mutated genome for each seed, this program can be skipped, but this has to be specified when running *OutbreakSimulator* (Chapter 3). Other than that, similar to *NetworkGenerator*, *SeedGenerator* should be executed either the mutation file will be provided by the users or will be provided by the burn-in process in *SeedGenerator*.

SeedGenerator will creates a new folder named `originalvcfs` under the provided working directory and stores one VCF file for each seed in the `originalvcfs` folder. Please note that `originalvcfs` folder will be removed and recreated if the existence of this folder is detected when running *SeedGenerator*, thus the old contents won't be preserved. A typical command to run *SeedGenerator* is:

```
python seed_generator.py \
    -wkdir ${YOUR_WORK_DIR} \
    -seed_size ${SEED_SIZE} \
    -method ${METHOD} \
    [Other params]
```

Listing 2.9: SeedGenerator Usage

We are now introducing all the options:

- `wkdir`: [*REQUIRED] (`string`)
  Absolute path to the working directory, which should typically stays consistent for each program being run.

- `seed_size`: [*REQUIRED] (`integer`)
  Number of the seeds you want to model, i.e. Number of infected hosts at the start of the simulation.

- `method`: [*REQUIRED] (`string`)
  The method of generating the seeds, which can only be one of the three methods: `user_input` or `SLiM_burnin_WF` or `SLiM_burnin_epi`. By specifying `user_input`, you are required to provide your own `.vcf` file by `-seed_vcf`. By specifying one of the SLiM-burnin methods, you are required to provide a reference genome by `ref_path` and other corresponding parameters for the method you choose.

- `seed_vcf`: [*OPTIONAL] (`string`)
  Required if `method` is specified as `user_input`. Full path to the seeds' vcf file user provided.

- `path_seeds_phylogeny`: [*OPTIONAL] (`string`)
  Can be specified only if `method` is specified as `user_input`, but not required. Full path to the seeds' phylogeny in a `.nwk` format.

- `ref_path`: [*OPTIONAL] (`string`)
  Required if `method` is specified as `SLiM_burnin_WF` or `SLiM_burnin_epi`. Full path to the reference genome of the pathogen you want to simulate, which has to stay consistent when running *OutbreakSimulator*.

- `mu`: [*OPTIONAL] (`float`)
  Required if `method` is specified as `SLiM_burnin_WF` or `SLiM_burnin_epi`. Mutation rate of the genomes during burn-in, which is the expected number of mutations per base pair for each genome in one tick. This doesn't have to stay consistent when running *OutbreakSimulator*, but please note that the branch length of phylogeny of seeds generated by *SeedGenerator* will be in the unit of tick, while the branch length of the transmission tree generated by *OutbreakSimulator* will be in the unit of tick as well, and the difference of mutation rate won't be shown in their branch length, which might not be desired for a lot of cases. Manual re-scaling of the seeds' phylogeny might be of people's interest after running *SeedGenerator* in this case.

- `n_gen`: [*OPTIONAL] (`int`)
  Required if `method` is specified as `SLiM_burnin_WF` or `SLiM_burnin_epi`. Number of ticks that will be run during the burn-in. All decisions of events including reproduction will only be executed once per tick, thus the final seeds' phylogeny will have a tree height at most being the number specified by `n_gen`.

- `Ne`: [*OPTIONAL] (`int`)
  Required if `method` is specified as `SLiM_burnin_WF`. The effective population size of a Wright-Fisher model.

- `host_size`: [*OPTIONAL] (`int`)
  Required if `method` is specified as `SLiM_burnin_epi`. Host population size, which needs to be the host size for the `contact_network.adjlist` file in the working directory provided by `-wkdir`.

- `seeded_host_id`: [*OPTIONAL] (`list of int`)
  Required if `method` is specified as `SLiM_burnin_epi`. ID(s) of the host(s) that will be infected with a pathogen having the reference genome at first tick of the burn-in process, which needs to be one or more of the host ids from `contact_network.adjlist` file in the working directory provided by `-wkdir`, i.e. chosen from 0 to host_size - 1.

- `S_IE_rate`: [*OPTIONAL] (`float`)
  Required if `method` is specified as `SLiM_burnin_epi`. The probability of a successful transmission between each infected host and each of its connected host that is susceptible each tick.

- `E_I_rate`: [*OPTIONAL] (`float`)
  Can be specified only if `method` is specified as `SLiM_burnin_epi` and that `-latency_prob` is also specifed to be bigger than 0. The probability of an exposed host becoming infectious each tick. The default value is 0, a value bigger than 0 is not required but recommended in this case, as exposed hosts won't activate (become infectious) if not specified, if `E_R_rate` is also not specified, then all hosts that became exposed will just take up a host's spot and never change the status, leading to mostly undesirable outcomes.

- `E_R_rate`: [*OPTIONAL] (`float`)
  Can be specified only if `method` is specified as `SLiM_burnin_epi` and that `-latency_prob` is also specifed to be bigger than 0. The probability of an exposed host recovering each tick. The default value is 0, meaning that exposed hosts won't recover.

- `latency_prob`: [*OPTIONAL] (`float`)
  Can be specified only if `method` is specified as `SLiM_burnin_epi`. The probability of a host becoming exposed upon being infected in the current generation, should be a number between 0 and 1. The default value is 0, meaning that all hosts become infectious upon infected.

- `I_R_rate`: [*OPTIONAL] (`float`)
  Can be specified only if `method` is specified as `SLiM_burnin_epi`. The probability of an infected host recovering each tick. The default value is 0, a value bigger than 0 is not required but recommended to be specified, or the infectious hosts will never change the status, leading to mostly undesirable outcomes.

- `I_E_rate`: [*OPTIONAL] (`float`)
  Can be specified only if `method` is specified as `SLiM_burnin_epi`. The probability of an infected host deactivating (going to exposed) each tick. The default value is 0, meaning that infected hosts won't deactivate to the exposed state.

- `R_S_rate`: [*OPTIONAL] (`float`)
  Can be specified only if `method` is specified as `SLiM_burnin_epi`. The probability of a recovered host losing immunnity (going back to susceptible) each tick. The default value is 0, meaning that infected hosts won't deactivate to the exposed state. A value bigger than 0 is not required but often desirable to be specified, as seeds' sequences will only be sampled in last tick of the burn-in, and in many cases the outbreak dies out before that tick is reached by default. With an appropriate `R_S_rate` specification, an endemic stage will have already been reached at last tick.

2.2.1 User input

If users want to provide their own seeds' mutation information, users need to make sure the provided file is a proper VCF format. The `.vcf` file should contain the exact number of samples as what is provided in `-seed_size`, and the sample names doesn't matter, as the seeds will be numbered based on the order in the provided `.vcf` file from 0 to seed_size − 1. If the provided `.vcf` file has more samples than what is specified in `-seed_size`, *SeedGenerator* will only take the first `${SEED_SIZE}` columns as seeds' genetic information. Please note that since EnivolCrossing only supports haploid pathogens for now, the phasing won't be informative in the `.vcf` file user provided. i.e. `0/1` and `1/0` or `1/1` in the sample columns will be all rewritten as `1` in the vcf files generated in the working directory provided by `-wkdir`. A typical command to run this mode is:

```
python seed_generator.py \
    -wkdir ${YOUR_WORK_DIR} \
    -seed_size ${SEED_SIZE} \
    -method user_input \
    -path_seeds_phylogeny ${SEED_PHYLO_PATH}
```

Listing 2.10: SeedGenerator Usage user_input

SeedGenerator will creates a new folder named `originalvcfs` under the provided working directory, and rewrite one `.vcf` file for each seed in the `${YOUR_WORK_DIR}/originalvcfs` directory, which will be named as `seed.X.vcf` where $X \in \{0, \cdots, \text{SEED\_SIZE}\}$. If `-path_seeds_phylogeny` is specified, user should make sure that the provided file is in a proper NWK format, where tip labels of the tree should be numbers from 0 to seed_size − 1, corresponding to the seeds' ID. *SeedGenerator* will read and check the provided file and rewrite it to the working directory, named as `seeds.nwk`. Please note that when *OutbreakSimulator* is run, it will detect the existense of `seeds.nwk` under working directory. If it is detected, then a whole transmission tree which seeds' phylogeny and transmission tree are concatenated into will be produced, please see more details in Chapter 3.

2.2.2 burn-in by Wright-Fisher model

To generate some pathogen genomes from a population with some desired genetic diversity. This is the option to run a burn-in process using a Wright-Fisher model, using SLiM back-end. the WF model will be run for a specified number of ticks specifed by `-n_gen` in a constant population size specified by `-Ne`. To run this mode, a typical command is:

```
python seed_generator.py \
    -wkdir ${YOUR_WORK_DIR} \
    -seed_size ${SEED_SIZE} \
    -method SLiM_burnin_WF \
    -ref_path ${REF_PATH} \
    -mu ${MU} \
    -Ne ${Ne} \
    -n_gen ${NUM_TICKS_BURNIN}
```

Listing 2.11: SeedGenerator Usage SLiM_burnin_WF

At the last tick in *SeedGenerator*, seeds' sequences will be sampled from the current population. *SeedGenerator* creates a new folder named `originalvcfs` under the provided working directory, and rewrite one `.vcf` file for each seed in the `${YOUR_WORK_DIR}/originalvcfs` directory, which will be named as `seed.X.vcf` where $X \in \{0, \cdots, \text{SEED\_SIZE}\}$. *SeedGenerator* records the phylogeny of the seeds sampled, but they do not always coalesce (i.e. can all be traced back to the same ancestor). If they do coalesce, the seeds' phylogeny will be written to the working directory, named as `seeds.nwk` in NWK format. If they don't coalesce, i.e. having multiple roots, *SeedGenerator* create a folder named `seeds_phylogeny_uncoalesced` in the working directory, and write down seeds' phylogeny for each root respectively in `.nwk` format in `${YOUR_WORK_DIR}/seeds_phylogeny_uncoalesced`.

For examples, if we want to run a burn-in process for COVID-19 genomes in WF model, we download COVID-19 genome from ncbi into the current directory, and the file is named `EPI_ISL_402124.fasta`. We specify the mutation rate being 1×10^{-6}, which corresponds the mutation rate of COVID-19 per day in real-time scale. We can run the burn-in for 3650 ticks, which could makes us simulating 10 years' burn-in period in real-time scale. We can use an effective population size of 1000, then we want to sample 5 seeds from the last tick, we can run:

```
python seed_generator.py \
    -wkdir ${YOUR_WORK_DIR} \
    -seed_size 5 \
    -method SLiM_burnin_WF \
    -ref_path EPI_ISL_402124.fasta \
    -mu 1e-6 \
    -Ne 1000 \
    -n_gen 3650
```

Listing 2.12: SeedGenerator Usage SLiM_burnin_WF example

2.2.3 burn-in by epidemiological model

Sometimes a burn-in process using the epidemiological model (which can be the same model and parameters as the real simulation) is desirable. in this case, *NetworkGenerator* program MUST has already been run. User need to specify an epidemiological model to do the burn-in. This mode can be actually viewed as a minimal version of the *OutbreakSimulator*, with sampling only permitted in the last tick, and no genetic effects. To run this mode, a typical command is:

```
python seed_generator.py \
    -wkdir ${YOUR_WORK_DIR} \
    -seed_size ${SEED_SIZE} \
    -method SLiM_burnin_epi \
    -ref_path ${REF_PATH} \
    -mu ${MU} \
    -n_gen ${NUM_TICKS_BURNIN} \
    -host_size ${HOST_SIZE}  \
    -seeded_host_id ${seeded_host_id} \
    -S_IE_rate ${S_IE_rate} \
    -E_I_rate ${E_I_rate} \
    -E_R_rate ${E_R_rate} \
    -latency_prob ${latency_prob} \
    -I_R_rate ${I_R_rate} \
    -I_E_rate ${I_E_rate} \
    -R_S_rate ${R_S_rate}
```

Listing 2.13: SeedGenerator Usage SLiM_burnin_epi

Note that not all the rates are required, but some are recommended for the compartmental model user tries to specify. Please refer to the simulator chapter for the full provided compartmental models and permitted transitions. You often want to avoid letting any state to be a dead end since that would cause an epidemic dying out. But it's not un-doable, be careful about the model you are specifing. If not enough pathogens exist in the last tick when seeds should be drawn, an error message will be popped out to instruct you to rerun the burn-in process or modify the parameters. The SEIR trajectory during the burn-in process is recorded and written as a compressed `.csv` file in the working directory provided by `-wkdir`, so users are welcome to inspect the file (named as `burn_in_SEIR_trajectory.csv.gz`) to get an impression of the dynamics produced by your current rate parameter settings. If enough pathogens are present, *SeedGenerator* creates a new folder named `originalvcfs` under the provided working directory, and rewrite one `.vcf` file for each seed in the `${YOUR_WORK_DIR}/originalvcfs` directory, which will be named as `seed.X.vcf` where $X \in \{0, \cdots, \text{SEED\_SIZE}\}$.

Same as the `SLiM_burnin_WF` option, *SeedGenerator* records the phylogeny of the seeds sampled, but they do not always coalesce if you seeded more than 1 host. If they do coalesce, the seeds' phylogeny will be written to the working directory, named as `seeds.nwk` in NWK format. If they don't coalesce, i.e. having multiple roots, *SeedGenerator* create a folder named `seeds_phylogeny_uncoalesced` in the working directory, and write down seeds' phylogeny for each root respectively in `.nwk` format in the sub-folder.

For examples, if we want to run a burn-in process for COVID-19 genomes in epidemiological model, we download COVID-19 genome from ncbi into the current directory, and the file is named `EPI_ISL_402124.fasta`. We specify the mutation rate being 1×10^{-6}, which corresponds the mutation rate of COVID-19 per day in real-time scale. We can run the burn-in for 3650 ticks, which could makes us simulating 10 years' burn-in period in real-time scale. We've already run *NetworkGenerator*, so say we have this contact network with 10000 hosts, and we think there's a cool host that we want to start with, which has the ID 256. We want to specify an epidemiological model where each tick, each infected hosts infect each of its connected host by a probability of 0.03, and all infected hosts will be in a latent state upon infection. Since the latency period for COVID-19 is usually 6 days after infection, we would have a rate of activation being 0.16. Since the recovery period for COVID-19 is usually 28 days after infection, we would have a rate of activation being 0.035. The immunity to COVID-19 after recovery last for up to 6 months but will be highly protective only for the first 2 months, so we choose 2 months as the rate of losing immunity, so that we would have a rate of immunity loss being 0.018. We don't want to include the circumstances where infected hosts go back to latent, or latent hosts recover. Then we want to choose 5 seeds from the last tick, so that we could run

```
python seed_generator.py \
    -wkdir ${YOUR_WORK_DIR} \
    -seed_size 5 \
    -method SLiM_burnin_epi \
    -ref_path EPI_ISL_402124.fasta \
    -mu 1e-6 \
    -n_gen 3650 \
    -host_size 10000 \
    -seeded_host_id 256 \
    -S_IE_rate 0.003 \
    -E_I_rate 0.16 \
    -E_R_rate 0  # This doesn't need to be specified since default value is 0 \
    -latency_prob 1 \
    -I_R_rate 0.035 \
    -I_E_rate 0  # This doesn't need to be specified since default value is 0 \
    -R_S_rate 0.018
```

Listing 2.14: SeedGenerator Usage SLiM_burnin_epi example

In some cases you might be confused about specifying `-seeded_host_id`. You can manually check your contact network file to find a host that you think is good, or you can utilize *HostSeedMatch* program (Section 2.4) to find a specific host that satisfy your need.

2.3 GeneticEffectGenerator

Genetic effect refers to the fitness of pathogens induced by mutations in specific genetic regions in their genomes. EnivolCrossing supports two kind of fitness operation based on genetic effect, which are transmissibility and drug-resistance. The *GeneticEffectGenerator* program specifies the structure of the genetic elements in the pathogen genomes by effect sizes of mutations in those regions. This program can be skipped if users wish to run a simulation without any fitness component, but other than that this program needs to be executed either user want to provide their own genetic effect model or want to randomly generate from a genome annotation file.

GeneticEffectGenerator will create a `.csv` file in the working directory named `causal_gene_info.csv`, where each row represents one causative region, the first three columns represents names of the regions, starting positions of the regions, and ending positions of the regions, other columns represents the effect sizes for each trait of mutations in each region. Note that traits can be either transmissibility or drug-resistance, specifying by column names, and more than one traits for transmissibility and/or drug-resistance can be supported. In the *OutbreakSimulator*, one effect size set has to be chosen for transmissibility and drug-resistance respectively for each epoch at most.

GeneticEffectGenerator will also calculate the trait values for each trait for all seeds' mutation profiles based on the effect sizes, thus *SeedGenerator* is required to be run before running *GeneticEffectGenerator*. The calculated trait values will be stored in `${YOUR_WORK_DIR}/seeds_trait_values.csv`, which isn't a prerequisite for running *OutbreakSimulator*, but might be helpful to take a look at before running *HostSeedMatch* to choose appropriate host to take specific seeds.

A typical command to run *GeneticEffectGenerator* is:

```
python genetic_effect_generator.py \
    -wkdir ${YOUR_WORK_DIR} \
    -method ${METHOD} \
    [Other params]
```

Listing 2.15: GeneticEffectGenerator Usage

We are now introducing all the options:

- `wkdir`: [*REQUIRED] (`string`)
  Absolute path to the working directory, which should typically stays consistent for each program being run.

- `method`: [*REQUIRED] (`string`)
  The method of generating the effect size, which can only be one of the two methods: `user_input` or `randomly_generate`. By specifying `user_input`, you are required to provide your own effect size file by `-effsize_path`. By specifying `randomly_generate`, you are required to provide number of traits you want to generate (`-trait_n`) and other parameters.

- `effsize_path`: [*OPTIONAL] (`string`)
  Required if `method` is specified as `user_input`. Full path to the effect size file user provided.

- `gff`: [*OPTIONAL] (`string`)
  Required if `method` is specified as `randomly_generate`. Full path to the GFF like file that stores the annotation of the genomes.

- `trait_n`: [*OPTIONAL*] (`list of int`)
  Required if `method` is specified as `randomly_generate`. How many different sets of effect sizes for transmissibility and drug-resistance is needed, thus has to be two integer numbers.

- `causal_size_each`: [*OPTIONAL] (`list of int`)
  Required if `method` is specified as `randomly_generate`. How many causal genetic regions should be chosen from the `.gff` file for each trait (transmissibility and drug resistance), thus has to be an integer list that has the same length of the sum of `-trait_n`.

- `es_low`: [*OPTIONAL] (`list of numeric`)
  Required if `method` is specified as `randomly_generate`. Lower bound of the effect size of each genetic region for each trait (transmissibility and drug resistance), thus has to be a numeric list that has the same length of the sum of `-trait_n`.

- `es_high`: [*OPTIONAL] (`list of numeric`)
  Required if `method` is specified as `randomly_generate`. Upper bound of the effect size for each trait (transmissibility and drug resistance), thus has to be a numeric list that has the same length of the sum of `-trait_n`.
- `normalize`: [*OPTIONAL] (`binary`)
  Default is `false`. Whether to normalize the effect sizes based on the length of the simulation and the mutation rate.
- `sim_generation`: [*OPTIONAL] (`list of numeric`)
  Required if `method` is specified as `randomly_generate` and `normalize` is specified as `true`. Number of ticks that is expected to be run in *OutbreakSimulator*.
- `mut_rate`: [*OPTIONAL] (`list of numeric`)
  Required if `method` is specified as `randomly_generate` and `normalize` is specified as `true`. Mutation rate that is expected to be used in *OutbreakSimulator*.

2.3.1 User input

If users want to provide their effect size file, users need to make sure the provided file is a proper format with identifiable column names. The template can be found at EnivolCrossing by Kim Lab. A typical command to run this mode is:

```
python genetic_effect_generator.py \
    -wkdir ${YOUR_WORK_DIR} \
    -method user_input \
    -effsize_path ${EFF_SIZE_PATH}
```

Listing 2.16: GeneticEffectGenerator Usage user_input

GeneticEffectGenerator will read the user-provided effect size file and check the format and entry of the file, then rewrite it to a file named `causal_gene_info.csv` in the working directory provided by `-wkdir`. Existence of this file in working directory is a prerequisite for running *OutbreakSimulator* when any of the genome-based transmissiibility and drug-resistance is activated.

2.3.2 Generate from GFF file

Sometimes users want to randomly generate an effect size file. A genome annotation file for the reference genome that user want to use in *OutbreakSimulator*, which can be downloaded from NCBI. Please make sure that all entries in the `.gff` files do not overlap in terms of genomic positions, or error may happens. Users can also divide the genome to arbitrary regions, but it has to be in a `.gff`-like format.

For transmissibility and drug-resistance, user could specify any number of sets of traits. A trait set is a set of effect sizes for the genetic regions selected, for each epoch in *OutbreakSimulator*, which set to use for transmissibility and drug-resistance has to be chosen. For example, two sets of drug-resistance effect sizes could be specified, representing resistance for two different type of drugs. In *OutbreakSimulator*, user could specify two epochs with treatment of different drugs by choosing different sets of effect sizes for drug-resistance.

For each trait set, a number of genetic regions, and lower bound and upper bound needs to be set, and the effect size will be uniformly drawn between the two bounds for each genetic region. The genetic regions will be randomly sampled from the `.gff` file provided.

If `normalize=true`, the effect sizes will be re-scaled based on the mutation rate and number of epochs provided in that the expected relative additive average effect size for the pathogen genomes at the end of the simulation will be 1. A typical command is:

```
python genetic_effect_generator.py \
    -wkdir ${YOUR_WORK_DIR} \
    -method randomly_generate \
    -gff ${GFF_PATH} \
    -trait_n ${NUM_SET_TRANSMISSIBILITY} ${NUM_SET_DRUGRESIST} \
    -causal_size_each ${NUM_GENE_SET_1} ... ${NUM_GENE_SET_N} \ # N = ${NUM_SET_TRANSMISSIBILITY}+${NUM_SET_DRUGRESIST}
    -es_low ${LOW_GENE_SET_1} ... ${LOW_GENE_SET_N} \
    -es_high ${HIGH_GENE_SET_1} ... ${HIGH_GENE_SET_N}
    [other params when normalize = T]
```

Listing 2.17: GeneticEffectGenerator Usage randomly generate

For example, we want to generate a set of effect sizes for Tuberculosis. Tuberculosis has around 4000 genes, among which we want to choose 5 to be causative to higher transmissibility, and we want to choose 3 to be causative to higher drug-resistance. We want to have two sets of drug-resistance traits, so that we can simulate two different treatment schemes later. We want to generate effect size between 0 and 5 for transmissibility. For the first drug-resistance trait, we want the range to be smaller, so we choose 1 and 3. For the second drug-resistance trait, we want the range to be bigger, so that we might get very different effect sizes for different genetic regions, so we choose between 0.1 and 7. We want to normalize all the effect sizes, since we are running the simulation for 1000 ticks with a mutation rate of 4.4×10^{-8}, we don't want the transmissibility to inflate too fast. To run this, we download the `.gff` file from ncbi for TB genome, named `GCF_000195955.2_ASM19595v2_genomic.overlap.gff`. After removing overlapping regions in the file, we can execute:

```
python genetic_effect_generator.py \
    -wkdir ${YOUR_WORK_DIR} \
    -method randomly_generate \
    -gff GCF_000195955.2_ASM19595v2_genomic.overlap.gff \
    -trait_n 1 2 \
    -causal_size_each 5 3 3 \
    -es_low 0 1 0.1 \
    -es_high 5 3 7 \
    -normalize true \
    -sim_generation 1000 \
    -mut_rate 4.4e-8
```

Listing 2.18: GeneticEffectGenerator Usage randomly generate example

The generated causal gene information file can be inspected and manually modified by users as well, but it's recommended to rerun by `user\_input` mode after modifications, so that traits values for each seed can be recalculated, and the format can be checked by the program. Please note that it's possible that one genetic regions has non-zero effect sizes for more than one trait, which provide a framework to simulate **pleiotropy**. This can't be deliberately specified in random generate mode, but it's a possible outcome. You can always manually change the effect sizes for specific purposes. The effect sizes can also be negative to simulate a negative effect to some traits. Like it's convenient to define a gene that mutations in it induced lower transmissibility and higher drug-resistance at the same time.

2.4 HostSeedMatcher

After determining the genetic architectures of the pathogen genomes, the host seed matching process is conducted to introduce index cases into the susceptible population. Upon successful execution of the program, users will receive a `seed_host_match.csv` file in the specified working directory. Below is a detailed guide on how to utilize the *HostSeedMatch* module:

```
python seed_host_match_func.py \
    -wkdir ${YOUR_WORK_DIR} \
    -method ${METHOD} \
    -n_seed ${NUMBER_OF_SEEDS} \
    [Other params]
```

Listing 2.19: HostSeedMatch Usage

The available options are as follows:

- `wkdir`: [*REQUIRED] (`string`)
  Absolute path to the working directory, which should be consistent with the directory used for running previous programs. It is assumed that the `contact_network.adjlist` file resides in this directory.
- `method`: [*REQUIRED] (`string`)
  The method used for conducting the matches, which can only be one of the two methods: `user_input` or `randomly_generate`. When specifying `user_input`, users are required to provide their own matching file using the `-path_matching` option. On the other hand, specifying `randomly_generate` necessitates providing matching methods for seeds (`match_scheme`) and corresponding matching parameters (`match_scheme_param`).
- `n_seeds`: [*REQUIRED] (`int`)
  Total number of seeds to be matched to the hosts.
- `path_matching`: [*OPTIONAL] (`string`)
  Absolute path to the user-provided matching file. The file must be a `.csv` file with two columns named `host_id` and `seed`, with each row corresponding to a pair of seed-host to be matched. This option is only required when `method` is set to `user_input`.
- `match_scheme`: [*OPTIONAL] (`JSON string` or `string`)
  A dictionary specifying the matching method for each seed. The values should be one of `["ranking", "percentile", "random"]`. If all the seeds are to be matched randomly, it can be "`random`". This option is only REQUIRED when `method` is set to `randomly_generated`.
- `match_scheme_param`: [*OPTIONAL] (`JSON string`)
  A dictionary containing parameters for each matching method. Refer to the sections below for the format of parameters corresponding to each matching method. This option is only REQUIRED when `method` is set to `randomly_generated`.

2.4.1 User-provided matching file input

Users have the option to match the seeds with hosts themselves without executing the *HostSeedMatch* module. However, it is highly recommended to run this program to ensure that the customized matching file meets the format requirements for subsequent simulation steps. Below is an example of how to check the format of the matching file:

```
python seed_host_matcher.py \
    -wkdir ${YOUR_WORK_DIR}$
    -method user_input
    -n_seed 10
    -path_matching ${PATH_TO_MATCHING_CSV_FILE}
```

Listing 2.20: HostSeedMatch user input example

2.4.2 Generate matching based on user-specified methods & parameters

If the preferred approach is to randomly assign all seeds within the population, a convenient shortcut command can be utilized to achieve this objective without the need to specify `match_scheme` or `match_scheme_param` options.

```
python seed_host_matcher.py \
    -wkdir ${YOUR_WORK_DIR} \
    -n_seed ${NUMBER_OF_SEEDS} \
    -method randomly_generated \
```

Listing 2.21: HostSeedMatch random matching example

Alternatively, the program offers automatic seed matching capabilities, allowing users to match seeds to hosts based on user-specified methods and parameters. Hosts are sorted in reverse order based on their number of contacts in the host contact network.

Ranking Method (`'ranking'`)

- Requires an integer to select the host of a given rank.
- Nodes with higher degrees have lower ranks.

Percentile Method (`'percentile'`)

- Requires a list of two integers within the closed interval [1, 100].
- Randomly selects a host within the specified percentile.
- Hosts with lower percentiles have higher connectivities.

Random Method (`'random'`)

- No parameter required, and any provided parameter will be ignored.
- Randomly selects a host from the susceptible population.

Consider the following scenario for a concrete example: suppose there is a need to conduct a matching for three seeds. For the first seed, the specific host it is matched to is inconsequential. However, for the second seed, it is desired to match it with a host possessing the highest degree of connectivity within the network. The third seed should be paired with a host from the latter half of the population, based on their connectivity rankings. To achieve this customized instance of matching, the following command would be executed:

```
python seed_host_matcher.py \
    -wkdir ${YOUR_WORK_DIR} \
    -n_seed 3 \
    -method randomly_generated \
    -match_scheme '{"0": "random", "1": "ranking", "2": "percentile"}'
    -match_scheme_param '{"1": 1, "2": [50, 100]}'
```

Listing 2.22: HostSeedMatch user specified matching method(s) example

Part II

Simulator

Chapter 3

Overview of the usage

Contents

3.1 Run OutbreakSimulator 29
3.2 Configuration file 30

3.1 Run OutbreakSimulator

In this part, we will introduce the *OutbreakSimulator* program of EnivolCrossing. To run this program, users should make sure that they already run all the programs in PART I (Chapter 1). As *OutbreakSimulator* will depend on the existence of several files and folders **in the working directory** specified:

- `contact_network.adjlist`: Required. Generated by *NetworkGeneratror* (Section 2.1)
- `seed_host_match.csv`: Required. Generated by *HostSeedMatch* (Section 2.4)
- `causal_gene_info.csv`: Required if user would like to use any genetic effect to modify fitness in their model. Generated by *GeneticEffectGenerator* (Section 2.3)
- `originalvcfs`(and the `seed.X.vcf` inside): Required if user would like to use a seed sequence different from the reference genome. Generated by *SeedGenerator* (Section 2.2)
- `seeds.nwk`: Optional. Generated by *SeedGenerator* (Section 2.2) If existing, *OutbreakSimulator* can generate the full phylogeny by binding the transmission tree to the seeds' phylogeny.

After making sure that these files exist per user need, the only input `OutbreakSimulator` needs is a configuration file that specifies all the parameters of the simulation. The template to this configuration file is here. After modifying all the values in the template file per user need, to run `OutbreakSimulator`, users can just run:

```
    python outbreak_simulator.py \
        -config ${YOUR_SIM_CONFIG_PATH}
```

Listing 3.1: OutbreakSimulator usage

- `config` [*REQUIRED] (`string`)
  Absolute path to the configuration file of running *OutbreakSimulator*.

We will talk about the structures and the parameters in the JSON file in this chapter. In next Chapter (Chapter 4) of this part, we'll introduce the modules of `OutbreakSimulator` and how the simulation actually works. In the last chapter of this part, we will walk you through some examples about how to simulate your own model (Chapter 5).

3.2 Configuration file

There are several sections in the configuration file: `"BasicRunConfiguration"`, `"EvolutionModel"`, `"SeedsConfiguration"`, `"GenomeElement"`, `"NetworkModelParameters"`, `"EpidemiologyModel"` and `"Postprocessing_options"`. Please note that this configuration file (Template config) is essentially a subset of the full configuration file (short template). The full config file is needed if you want to run streamlined (Part IV).

3.2.1 BasicRunConfiguration

This section specifies basic configuration to run the simulator.

- `"cwdir"`: (`string`) Absolute path to the working directory, which should be the path that you used to run all the pre-simulation programs.
- `"n_replicates"`: (`int`) Using the same set of configurations (specified by this file), how many replicates do users want to run. (`OutbreakSimulator` will create sub-directories for outputs of each replication in the working directory specified by `"cwdir"`).

3.2.2 EvolutionModel

This section specifies evolutionary parameters and fitness calculation modes used in the simulation.

- `"n_generation"`: (`int`) How many ticks user want to run in the simulation.
- `"mut_rate"`: (`float`) A uniform mutation rate (μ) across the genome. The value represents the expected number of mutations happening at one position each tick for each pathogen genome. `OutbreakSimulator` uses the Jukes-Cantor model in SLiM, thus, the parameter α for the Jukes-Cantor model is essentially $\mu/3$.
- `"trans_type"`: (`string`) Model for calculating transmissibility. Two types are provided: `"additive"` and `"bialleleic"`, please refer to Section 4.2 as details of the two models.
- `"dr_type"`: (`string`) Model for calculating drug-resistance. Two types are provided: `"additive"` and `"bialleleic"`, please refer to Section 4.2 as details of the two models.
- `"within_host_reproduction"`: (`binary`) `true` or `false`. Whether to activate within-host reproduction (??).
- `"within_host_reproduction_rate"`: (`float`) Probability of reproducing (branching event) within-host for each existing pathogen genome each tick. Won't be used unless `"within_host_reproduction": true`.
- `"cap_withinhost"`: (`int`) The maximum number of pathogens that is allowed to be existing in one host at the same tick. i.e. if the cap is reached, then no within-host reproduction or super-infection could happen for this host. Should be 1 if no within-host reproduction or super-infection is activated.

3.2.3 SeedsConfiguration

This section specifies the sequences of the seeds.

- `"seed_size"`: (`int`) Number of seeds. Should be the same as what is used for *SeedGenerator* and *HostSeedMatcher* unless `"use_reference": true`, in which case should be the same as what is used for *HostSeedMatcher*.
- `"use_reference"`: (`binary`) `true` or `false`. Whether to use reference genome as seed sequences. If `true`, then *SeedGenerator* won't be needed to run ahead of time. If `false`, then *SeedGenerator* should have been run.

3.2.4 GenomeElement

This section specifies the genetic elements settings that is going to be used in the simulation.

- `"use_genetic_model"`: (`binary`) `true` or `false`. Whether to use genetic model to calculate fitness (transmissibility and drug-resistance). If `false`, then *GeneticEffectGenerator* won't be needed to run ahead of time. If `true`, then *GeneticEffectGenerator* should have been run.
- `"ref_path"`: (`string`) Absolute path to the reference genome in `.fasta` format. Should be the same that is been used for *SeedGenerator* if applicable.
- `"traits_num"`: (`list of ints`) A list of length 2, number of effect size sets for transmissibility and drug-resistance respectively. e.g. `"traits_num":[1,2]` represents 1 set for transmissibility and 2 sets for drug-resistance, should be the same as what is used in *GeneticEffectGenerator*'s `-trait_n` option.

3.2.5 NetworkModelParameters

This section specifies the host population information that is going to be used in the simulation.

- `"use_network_model"`: (`binary`) `true` or `false`. Whether to use a host contact network for underlining transmission network. If `false`, then *GeneticEffectGenerator* won't be needed to run ahead of time and a random branching process will be run instead (STILL UNDER DEVELOPMENT). If `true`, then *GeneticEffectGenerator* should have been run.
- `"host_size"`: (`int`) host population size, should be the same as what has been used in *NetworkGenerator*.

3.2.6 EpidemiologyModel

This section specifies the epidemiological model and all the transitions between them.

- `"model"`: (`string`) The compartmental model that will be used, choose from `"SIR"` or `"SEIR"`. See Section 4.3 for the permitted compartments and transitions.
- `"epoch_changing"`: A subsection that specifies the epoch property of the simulation. Please see more in Section 4.4.
  - `"n_epoch"`: (`int`) Number of epochs in the simulation. All the parameters in `"genetic_architecture"` and `"transiton_rate"` section will be lists of the length of `n_epoch`, as one set of rates needs to be provided for each epoch.

  - `"epoch_changing_generation"`: (`list of int`) Which tick(s) should the configuration be moved to the next epoch. Should be a list of the length of `n_epoch - 1`. For examples, if `"n_epoch": 3` is specified, then you should specify `"epoch_changing_generation":[t1, t2]`, where $t1, t2 \in \{1, \cdots, \text{n\_generation}\}$ (`"n_generation"` was specified in Section 3.2.2)

- `"genetic_architecture"`: A subsection that specifies the genetic architecture to be used in each epoch.
  - `"transmissibility"`: (`list of int`) A list of which effect size set should be used for transmissibility for each epoch. Should be a list having the length of `n_epoch`, and each element being an integer representing the trait id. For example, in `"traits_num"` you specified `[2,3]`, meaning that you have 2 effect size sets for transmissibility, then here each element in `"transmissibility"` should be one of $\{0, 1, 2\}$, where 0 represents not using genetic effect on transmissibility in this epoch, and 1 or 2 represent the set of effect size that will be used in this epoch.
  - `"cap_transmissibility"`: (`list of float`) The cap of transmissibility values in this epoch. Due to the nature of our simulation, transmissibility can inflate if all effect sizes are positive. You can set a cap for each epoch so that if transmissibility is calculated to be higher than the cap, then the cap value is being used. Should be a list having the length of `n_epoch`.
  - `"drug_resistance"`: (`list of int`) A list of which effect size set should be used for drug-resistance for each epoch. Should be a list having the length of `n_epoch`, and each element being an integer representing the trait id. For example, in `"traits_num"` you specified `[2,3]`, meaning that you have 3 effect size sets for drug-resistance, then here each element in `"transmissibility"` should be one of $\{0, 1, 2, 3\}$, where 0 represents not using genetic effect on drug-resistance in this epoch, and 1 or 2 or 3 represent the set of effect size that will be used in this epoch.
  - `"cap_drugresist"`: (`list of float`) The cap of drug-resistance values in this epoch. Due to the nature of our simulation, drug-resistance can inflate if all effect sizes are positive. You can set a cap for each epoch so that if drug-resistance is calculated to be higher than the cap, then the cap value is being used. Should be a list having the length of `n_epoch`.

- `"transition_rate"`: A subsection that specifies the (basic) transition probability between compartments in each epoch.
  - `"S_IE_rate"`: (`list of float`) The basic probability of a successful transmission between each infected host and each of its connected host that is susceptible every tick for each epoch. Should be a list having the length of `n_epoch`, and the actual probability could be modified by transmissibility if `"transmissibility"` isn't 0 in this epoch.
  - `"I_R_rate"`: (`list of float`) The probability of an infected host recovering every tick for each epoch. Should be a list having the length of `n_epoch`. In epochs where `"drug_resistance"` isn't 0, this is essentially the treatment probability before resistance component.
  - `"R_S_rate"`: (`list of float`) The probability of a recovered host losing immunity (going back to susceptible) every tick for each epoch. Should be a list having the length of `n_epoch`.
  - `"latency_prob"`: (`list of float`) The probability of a host becoming exposed upon being infected in the current tick for each epoch. Should be a list having the length of `n_epoch`.
  - `"E_I_rate"`: (`list of float`) The probability of an exposed host becoming infectious every tick for each epoch. Should be a list having the length of `n_epoch`.
  - `"I_E_rate"`: (`list of float`) The probability of an infected host becoming latent (exposed) every tick for each epoch. Should be a list having the length of `n_epoch`.

  - `"E_R_rate"`: (`list of float`) The probability of an exposed host recovering every tick for each epoch. Should be a list having the length of `n_epoch`.
  - `"sample_rate"`: (`list of float`) The probability of an infected host being sampled every tick for each epoch. Should be a list having the length of `n_epoch`.
  - `"recovery_prob_after_sampling"`: (`list of float`) The probability of an sampled host recovering upon sampling every tick for each epoch. Should be a list having the length of `n_epoch`.

- `"massive_sampling"`: A subsection that specifies massive sampling events (if any). Please refer to Section 4.5 for how does these events work.
  - `"event_num"`: (`int`) Number of massive sampling events.
  - `"generation"`: (`list of int`) Which ticks should each massive sampling event happening. Should be a list having the length of `event_num`.
  - `"sampling_prob"`: (`list of float`) Probability of infected hosts being sampled in each massive sampling event. Should be a list having the length of `event_num`.
  - `"recovery_prob_after_sampling"`: (`list of float`) Probability of infected hosts recovering upon being sampled in each massive sampling event. Should be a list having the length of `event_num`.

- `"super_infection"`: (`binary`) `true` or `false`. Whether to permit super-infection. If `true`, then infected hosts could still be infected except for that it has already reaches the cap specified in `"cap_withinhost"`. And all simultaneous (in the same tick) infections to one host will be preserved.

- `"Postprocessing_options"`: A subsection that specifies whether and how post-simulation data processing is done. Please refer to Part III for all post-processing details.
  - `"do_postprocess"`: (`binary`) `true` or `false`. Whether to do post-simulation processing. If `true`, then all plots and data will be generated. If `false`, then for each replicate, only logged events, treesequence file of the samples and transmission trees for each seed will be generated.
  - `"tree_plotting"`: A sub-subsection about plotting transmission trees.
    - `"branch_color_trait"`: (`int`) How to color the branches in the generated tree plot, only useful if `"do_postprocess": true`. If specified as 0, branches will be colored by seed. If specified as ids of other traits (both transmissibility and drug-resistance are ordered together), branches will be colored by relative trait value (lowest being blue, highest being red).
    - `"drug_resistance_heatmap"`: (`int`) (`binary`) `true` or `false`. Whether to do plot the drug resistance values of each sample as heatmap in the transmission tree plot, only useful if `"do_postprocess": true` and there is drug-resistance traits.

Chapter 4

Modules of the simulator

Contents

4.1 Evolutionary model **35**
4.2 Genetic effects to fitness **35**
4.3 Compartmental model **36**
4.4 Epoch structure **37**
4.5 Massive sampling events **37**

4.1 Evolutionary model

In EnivolCrossing, evolutionary process is coupled with epidemiological process and they happen in the same time scale. As EnivolCrossing uses SLiM as the engine to run *OutbreakSimulator*, the mutation process also follows the fashion of SLiM in that they happens in a specified rate (`mut_rate`) when ”reproduction” happens. ”Reproduction” here represents the term defined in SLiM. In a typical SLiM non-WF model, reproduction and generations are not at the same scale as ticks. But EnivolCrossing is designed such that for every existing pathogen, reproducuction of themselves happens as the first events each tick, thus mutations are imposed at the same time scale as other epidemiological process. Old pathogens will be killed at the end of each tick, to ensure that only one copy of the same pathogen genome exists at the same time. Other reproduction events are transmission events and within-host reproduction events, governed by the model and parameters user specified. When a transmission event happens, a reproduction event happens for the pathogen in the infector host, and the offspring pathogen is planted into the infectee host. When a within-host reproduction event happens, the offspring is in the same host. The two events won’t happen when the receiver host has already reached the cap of pathogens.

4.2 Genetic effects to fitness

In EnivolCrossing, two types of fitness can be modeled by genetic effect, transmissibility and drug-resistance. Two different genetic architecture, additive and bialleleic could be set for transmission rate trait and treatment failure rate trait separately. By setting an additive architecture, the contribution of this region to the trait will be the sum of contribution of all existing mutations in this genetic region in the genome that is calculated, where each mutation has the same effect size as specified in the genetic architecture file. By setting a bialleleic genetic architecture, the contribution of this region to trait value will be a binary variable that takes exactly the effect size value if at least one mutation exists in this region, and takes the value 0 when this region is the same as the reference genome. The overall trait value of this pathogen genome is the sum of the contribution of all causal regions. The trait value of a genome i is calculated by

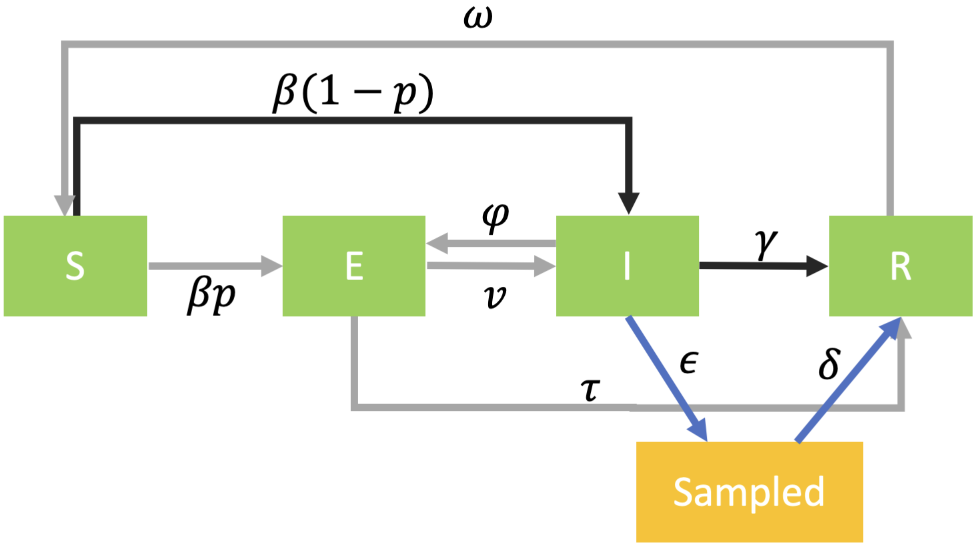
$$\text{Trait}_i = \Sigma_{j=0}^{n} q_j \times x_{ij}$$

where q_j represents the effect size of causal region j and x_{ij} represents the scaling factor for contribution of causal region j for genome i. x_{ij} equals to the number of mutations in causal region j for genome i in an additive architecture, and equals to $\mathbb{1}\{\text{There exists at least one mutation in causal region } j \text{ for genome } i\}$ in a bialleleic model.

4.3 Compartmental model

EnivolCrossing employs a classical compartmental epidemiological model by dividing all hosts into four compartments, susceptible, exposed, infected and recovered. The transition between the compartments have a basic rate that needs to be specified in the configuration file. EnivolCrossing supports a wide range of transitions.

In every tick, transmission events are decided first. For every pair of infected host i and a connected host j, a decision is made as to whether a transmission event happens. This is a bernoulli trial with parameter $\beta' = \beta \times \text{Transmissibility}_i \times \mathbb{1}\{j \text{ is available}\}$, where $\{j \text{ is available}\}$ means that the potential infectee host is susceptible or hasn't reached the cap of pathogen load if `superinfection=true`. β is the basic transmission probability (`S_IE_rate`), $\text{Transmissibility}_i$ is the transmissibility calculated by the effect size model specified for this epoch for one sampled pathogen from host i. See Section 4.2 for details.



In every tick, after the transmission event, hosts' states are going to be changed.

- For susceptible hosts that are just infected in the same tick, there states hasn't been decided. It's decided by a Bernoulli trial of parameter p (`latency_prob`) to see whether they are transit into exposed or infected states. These newly transit exposed and infected hosts won't take part in other transitions in this epoch. For other susceptible hosts, no transition will be made. If `superinfection=false`, for hosts that are infected more than once in this tick, only one of the infection events will be kept valid. Pathogens transmitted by other events will be removed.
- For exposed hosts, their new states are decided by a random draw from a multinoulli distribution of the parameter v, τ and $1 - \tau - v$, which represents the probability of activation (`E_I_rate`), recovery from exposed (`E_R_rate`) and staying exposed in this epoch.

- For infected hosts, their new states are decided by a random draw from a multinoulli distribution of the parameter γ, ϵ, ϕ and $1-\gamma-\epsilon-\phi$, which represents the probability of recovery (`I_R_rate`), being sampled (`sample_rate`), deactivation (`I_E_rate`) and staying infected in this epoch. For those chosen to be sampled, there's also a probability δ of recovery upon sampling (`recovery_prob_after_sampling`), if not recovering upon sampling, than the host stay infected. If treatment is turned on in this tick (the epoch that the tick is in uses `drug-resistance` different from 0), then all hosts that are experiencing recovery events in this tick needs to go through another random draw to decide whether the treatment failed (i.e. not recovering in this tick), see Section 4.2 for details.
- For recovered hosts, their new states are decided by a Bernoulli trial of parameter ω (`R_S_rate`) to see whether they lose their immunity.

4.4 Epoch structure

OutbreakSimulator is structured in epochs. Epoch is an era of the outbreak that has a set of parameter settings and span some consecutive ticks. By specifying more than one epochs, users will be responsible to provide more than one set of parameters, by which changing of environment can be modeled, like new policy and new treatment scheme being implemented. The epochs should span all the ticks, so user only needs to provide the tick when epochs are going to be changed, and *OutbreakSimulator* automatically uses the first epoch starting from first tick until first epoch-changing event, and use the last epoch untilk the end of the simulation.

4.5 Massive sampling events

In reality, a massive sampling effort in a short time frame often happens for pathogens. This process happens after all other state transitions are executed, and decide whether the remained infected hosts are sampled by the massive sampling effort, recovery rate upon massive sampling can also be defined. This by default doesn't happen, you need to specify which ticks the massive sampling happens, and each event should have a different set of parameters as well. The massively sampled pathogens and the normally sampled pathogens aren't distinguished in the final sample dataset.

Chapter 5

Specifying your own configuration

Contents

5.1 A minimal model for transmissibility 39
5.2 Multi-stage drug treatment 42

In this chapter, some examples of sampling different scenarios are provided to help understand how the configuration file and *OutbreakSimulator* works.

5.1 A minimal model for transmissibility

In this section, we will simulate a Tuberculosis outbreak in a host population of size $10,000$ for 10 years. The time scale is that we want each tick to represent 1 day, so we want to simulate 3650 ticks. The mutation rate for tuberculosis genome is 0.5 SNPs per genome per year, which scale to be 3.12×10^{-10} SNPs/bp/day.

1. The first step is to run *NetworkGenerator*. We want to simulate a well-mixed population with every individual having 10 contacts. We thus run

```
    python network_generator.py \
        -popsize 10000 \
        -wkdir ${YOUR_WKDIR} \
        -method randomly_generate \
        -model ER \
        -p_ER 0.001

```

Listing 5.1: NetworkGenerator model 1

2. We then run *SeedGenerator* to generate the seeds. We want to run a Write-Fisher model of $N_e = 1000$ and run for 4000 generations. During the burn-in, we want to let one generation to be scaled to 1 year, we thus use the mutation rate 1.1×10^{-7}. We want to generate 5 seeds. The reference genome we use here is Tuberculosis genome, which we put in our repository as well (TB genome).

```
    python seed_generator.py \
        -wkdir ${YOUR_WKDIR} \
        -seed_size 5 \
        -method SLiM_burnin_WF \
        -Ne 1000 \
        -ref_path GCF_000195955.2_ASM19595v2_genomic.fna \
        -mu 1.1e-7 \
        -n_gen 4000
```

```

```

Listing 5.2: SeedGenerator model 1

If you are lucky enough that the seeds do coalesce, you will see a `seeds.nwk` appear in your working directory, if you would like to scale the tree to based on mutation rate, manually modify the branch length to make them 365 times the original branch lengths. But now for demonstration, we will just leave the file as it be. If you don't see the `seeds.nwk`, it's fine but you won't get a full phylogeny tree at last, rerun this program until you get a coalesced result if you want.

3. Then we would like to run *GeneticEffectGenerator* using a `.gff` file. A `.gff` file for the TB genome that we downloaded is and modified also in our repository (TB gff). We also want to normalize the effect size based on the generation time we are running. We thus run

   ```
       python genetic_effect_generator.py \
           -wkdir ${YOUR_WKDIR} \
           -method randomly_generate \
           -trait 1 0 \
           -es_low 1 \
           -es_high 10 \
           -gff GCF_000195955.2_ASM19595v2_genomic.overlap.gff \
           -causal_size_each 5 \
           -normalize T \
           -mut_rate 3.12e-10 \
           -sim_generation 3650

   ```

   Listing 5.3: GeneticEffectGenerator model 1

   if you look at the working directory, two new files `causal_gene_info.csv` and `seeds_trait_values.csv` appears. Take a look at `seeds_trait_values.csv`, which shows the transmissibility value for each seed based on the genetic architecture that was generated and stored in `causal_gene_info.csv`. You can rerun this program or manually modify `causal_gene_info.csv` and run this program in `-method user_input` mode to see the seeds' trait value based on your modified genetic architecture as well.

4. Last step before running the simulation is that we want to match the seeds to hosts by *HostSeedMatcher*. This time we want all seeds to be matched randomly. So that we run:

   ```
       python seed_host_matcher.py \
       -wkdir ${YOUR_WKDIR} \
       -n_seed 5 \
       -method randomly_generate

   ```

   Listing 5.4: HostSeedMatcher model 1

   You will see a `seed_host_match.csv` appear in the working directory.

5. Now we are all good to run *OutbreakSimulator* To do that, we first download the template from our Github repo (short template). Then modify it based on your need. According to CDC, latency period for TB can be around 3 months, recovery time can be 4 months for rifapentine-moxifloxacin treatment regimen. We want to model that all new infections lead to latency at first. Though it's controversal, but several study indicates that immunity doesn't exist in TB. We thus assume a quick rate of losing immunity to be around 20 days. For basic transmission rate, we assume an arbitrary probability of 0.004, meaning that for each contact pair, an everyday infection probability is 0.4%. We don't want to model other transitions between compartments. And we don't want to model massive sampling events. This is the config file we specify using these parameters:

```
{
  "BasicRunConfiguration": {
    "cwdir": "/Users/px54/Documents/TB_software/V2_code/test/test_minimal_model",
    "n_replicates": 3
  },
  "EvolutionModel": {
    "n_generation": 3650,
    "mut_rate": 3.12e-10,
    "trans_type": "additive",
    "dr_type": "additive",
    "within_host_reproduction": false,
    "within_host_reproduction_rate": 0,
    "cap_withinhost": 1
  },
  "SeedsConfiguration": {
    "seed_size": 5,
    "use_reference": false
  },
  "GenomeElement": {
    "use_genetic_model": true,
    "ref_path": "/Users/px54/Documents/TB_software/V2_code/test/data/TB/GCF_000195955.2_ASM19595v2_genomic.fna",
    "traits_num": [1, 0]
  },
  "NetworkModelParameters": {
    "use_network_model": true,
    "host_size": 10000
  },
  "EpidemiologyModel": {
    "model": "SEIR",
    "epoch_changing": {
      "n_epoch": 1,
      "epoch_changing_generation": []
    },
    "genetic_architecture": {
      "transmissibility": [1],
      "cap_transmissibility": [10],
      "drug_resistance": [0],
      "cap_drugresist": [0]
    },
    "transiton_rate": {
      "S_IE_rate": [0.003],
      "I_R_rate": [0.008],
      "R_S_rate": [0.05],
      "latency_prob": [1],
      "E_I_rate": [0.01],
      "I_E_rate": [0],
      "E_R_rate": [0],
      "sample_rate": [0.00001],
      "recovery_prob_after_sampling": [0]
    },
    "massive_sampling": {
      "event_num": 0,
      "generation": [],
      "sampling_prob": [],
      "recovery_prob_after_sampling": []
    },
    "super_infection": false
  },
  "Postprocessing_options": {
    "do_postprocess": true,
    "tree_plotting": {
      "branch_color_trait": 1,
      "drug_resistance_heatmap": true
    }
  }
}

```

Listing 5.5: config model 1

We thus use absolute path to this configuration file to run *OutbreakSimulator* by:

```
    python seed_generator.py \
        -config ${PATH_TO_THIS_CONFIG}

```

Listing 5.6: OutbreakSimulator model 1

You should then see the simulator running, the steps it's currently in will be printed out in the standard output. After it finished, you can look into the working directory and each replicate's output directory for the results. The results of one test run are in our repository as well (minimal model).

5.2 Multi-stage drug treatment

To be finished.

Part III

Process the output data

Chapter 6

Output structures

Contents

6.1 Reading the output plots 45

6.1 Reading the output plots

Part IV

Streamline

Part V

GUI

Part VI

Examples