EnivolCrossing Manual

Perry Xu, Shenni Liang, Jae Hahn, Vivian Zhao

Update on May 3, 2024

Contents

Glossary

drug-resistance	The probability of treatment failure on each pathogen induced by its genome. 22, 32, 37
epoch	The simulation process can be divided into multiple epochs, during which all rate parameters and effect size traits can be set differently. 22, 34, 39
host	The larger organism that harbours one or more pathogens. 7, 33
initial sequences	The pathogen genome(s) that infect a fully susceptible host population as outbreak simulation starts. 16, 33
pathogen	The organism whose movements and genomes being modelled. 7
super-infection	The situation when a host is infected by a pathogen while still harboring another infection, either simultaneously or sequentially without full recovery from the initial infection. 32
tick	Taken from the **tick** defined in population genetics simulation framework SLiM. It is the time unit of the epidemiological events, the time unit of mutation, and the branch length unit of the transmission tree. 17, 32, 34
transmissibility	The relative transmission probability of each pathogen induced by its genome. 22, 32, 34, 37

Glossary

Preface

Contents

0.1 An Overview of EnivolCrossing

EnivolCrossing (**Ep**I**d**emiological 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 structure. EnivolCrossing highlights the coupling of evolutionary and epidemiological processes 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, 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 backend. By generating appropriate SLiM scripts per configuration, different epidemiological scenarios could be simulated, including various permissible transitions between compartments and 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
- License: Copyright

Part I

Pre-simulation programs

Chapter 1

Overview

Before running all the programs for the *OutbreakSimulator* of EnivolCrossing, it is essential to sequentially specify certain input files by following pre-simulation programs. Here are the required files for pre-simulation:

- A **reference genome** in `fasta` format

- A **working directory** that stays the same for running all programs (The 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, in order to randomly generate a genetic architecture, a `.gff`-like (general feature format) file will be needed when running *GeneticEffectGeneration* program. We will introduce all modes of each program and their respective required inputs below.

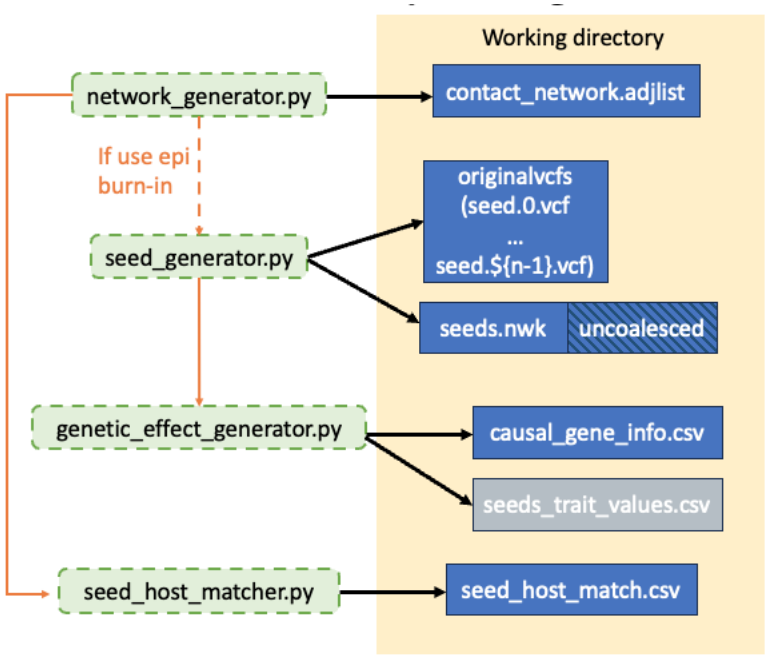
You should adhere to a specific workflow to ensure the proper functioning of dependent programs. Below is a typical sequence of running pre-simulation along with their dependencies:

1. Run *NetworkGenerator*. Output of this program might be the dependency files for *SeedGenerator* in some modes (e.g., burn in with epidemiology model).

2. Run *SeedGenerator* if the initial sequences are different from the provided reference genome in simulations. If you want to use reference genome as the initial sequence, then this step can be omitted.

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* to match each seed to a specific host.


Working directory
network_generator.py
contact_network.adjlist
If use epi burn-in
originalvcfs (seed.0.vcf ... seed.${n-1}.vcf)
seed_generator.py
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 one by one

Contents

2.1 NetworkGenerator

EnivolCrossing models the host population and their contact network explicitly. *NetworkGenerator* is used to generate an adjacency list in the working directory that stores the host contact network information. To run *NetworkGenerator*, the required inputs are working directory (`-wkdir`), host population size (`-popsize`) and method of network input (`-method`). Alternatively, you can provide a customized network instead of generating one, but you would still have to execute this program for sanity check on the network format. Here is how you can run *NetworkGenerator*:

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

Listing 2.1: NetworkGenerator Usage

We introduce all the options:

- `wkdir`: [*REQUIRED] (`string`)
  Absolute path to the working directory, which should be the same for all programs.

- `popsize`: [*REQUIRED] (`integer`)
  Size of the host population, i.e. Number of nodes in the network 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`. Absolute path to the customized contact network file. 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 apply for generating the contact network. The choices are `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 contact between two individuals.

- `rp_size`: [*OPTIONAL] (`list of ints`)
  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 contact between individuals within the same partition. As of now, we only support a two-partition model, thus this option has to be a float list of size 2, each representing a contact probability between individuals within 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 contact between individuals from different partitions.

- `m`: [*OPTIONAL] (`float`)
  Required if `model` is specifed as `BA`. It is the parameter for generating a Barabasi-Albert graph (Number of contacts to attach from a new individual to existing individuals when creating this network).

- `random_seed`: [*OPTIONAL](`int`)
  Optional parameter to make the result reproducible if desired. Using the same parameter ensures identical outputs when the scripts are rerun.

In the upcoming sections, we introduce logistics and examples about the generation of various host contact networks.

2.1.1 User input

If the user opts to supply their own contact network, they need to make sure the contact network file is in an adjacency list format, which is a space-delimited file. Lines starting with `#` will be marked as comments. A typical command to run this mode is:

```
python network_generator.py \
    -wkdir ${WKDIR} \
    -popsize ${HOST_SIZE} \
    -method user_input \
    -path_network ${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* cannot be run without a properly formatted and named `contact_network.adjlist` in the working directory. The step is recommended even if you're providing your own contact network. However, you can modify `contact_network.adjlist` by adding/deleting new contact edges manually as well, but we recommend rerunning *NetworkGenerator* by taking the absolute path of `${WKDIR}/contact_network.adjlist` as input for `-path_network` after your manual modification to make sure it is 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 will not 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 ${WKDIR} \
    -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 $10 (= 10000 * 0.001)$ other hosts, we run:

```
python network_generator.py \
    -wkdir ${WKDIR} \
    -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.

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

Listing 2.6: NetworkGenerator BA network example

2.1.4 Two Sub-populations

This option is designed for a host contact network with sub-population structures, where there are two partitions in the whole graph. Each partition has their own intra-partition contact probability and inter-partition contact probability. This sub-population configuration can represent the separate rural and urban area, two countries, ecological barrier in natural world, and so on. A typical command for this option is:

```
python network_generator.py \
    -wkdir ${WKDIR} \
    -popsize ${HOST_SIZE} \
    -method randomly_generate \
    -model RP \
    -rp_size ${POP1_SIZE} ${POP2_SIZE} \
    -p_within ${POP1_PROB} ${POP2_PROB} \
    -p_between ${PROB_BETWEEN_POP1_POP2}
```

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 example, to generate a uniform host population of 10000 hosts, of which 6000 hosts come from a urban area where expected connectivity for each host is high as $12 (= 6000 * 0.002)$, and the rest 4000 come from a rural area where expected connectivity for each host is as low as $4 (= 4000 * 0.001)$. Meanwhile, there are a few connections between the two area so that hosts from one partition are expected to be connected with $3 (= 6000 * 0.0005)$ hosts from the other partition, we run:

```
python network_generator.py \
    -wkdir ${WKDIR} \
    -popsize 10000 \
    -method randomly_generate \
    -rp_size 4000 6000 \
    -p_within 0.001 0.002 \
    -p_between 0.0005
```

Listing 2.8: NetworkGenerator RP network example

2.2 SeedGenerator

Initial sequences (a.k.a., seeds) are pathogen genomes that are planted to the host population at the start of the simulation. They might have distinct genomic sequences and different fitness in terms of transmissibility and drug-resistance related to specific mutations they harbour. If the user wants simulations to start from the original reference genome instead of a mutated genome for each initial sequences, this program can be skipped. As a side note, using the reference genome as the initial sequence has to be specified when running

OutbreakSimulator (Chapter 3). Other than that, *SeedGenerator* should be executed either when the mutation file is provided by the user or generated by the burn-in process in *SeedGenerator*.

SeedGenerator will create a new folder named `originalvcfs` under the provided working directory and stores one VCF file for each initial sequence in the `originalvcfs` folder. Please note that `originalvcfs` folder will be removed if it already exists when running *SeedGenerator*. As a result, the old contents from previous runs under the same working directory will be overwritten. To run *SeedGenerator* is:

```
python seed_generator.py \
    -wkdir ${WKDIR} \
    -num_init_seq ${NUM_INIT_SEQ} \
    -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 stay the same for each program.

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

- `method`: [*REQUIRED] (`string`)
  The method of generating the initial sequences, 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.

- `init_seq_vcf`: [*OPTIONAL] (`string`)
  Required if `method` is specified as `user_input`. Absolute path to the initial sequences' vcf file.

- `path_init_seq_phylogeny`: [*OPTIONAL] (`string`)
  Optional if `method` is specified as `user_input`. Absolute path to the initial sequences' phylogeny in a `.nwk` format.

- `ref_path`: [*OPTIONAL] (`string`)
  Required if `method` is specified as `SLiM_burnin_WF` or `SLiM_burnin_epi`. Absolute path to the reference genome of the pathogen you want to simulate with.

- `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. `mu` need not have the same value when running *OutbreakSimulator*, but please note that the branch length of phylogeny of initial sequences generated by *SeedGenerator* will be in the unit of tick, and the branch length of the transmission tree generated by *OutbreakSimulator* will be in the unit of tick as well. Hence, the difference of mutation rates will not be shown in their branch lengths, which might not be optimal if it is desirable to reflect the number of mutations on the transmission tree. Manual re-scaling of the initial sequences' 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/generations that will be run during the burn-in. All events including reproduction will only be executed once per generation, thus the initial sequences' phylogeny will have a tree height of at most `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 ints`)
  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 an infected host and each of their connected host that is susceptible at each tick.

- `E_I_rate`: [*OPTIONAL] (`float`)
  Required if `method` is specified as `SLiM_burnin_epi` and that `-latency_prob` is also specifed to be greater than 0. The probability of an exposed host becoming infectious during each tick. The default value is 0, a positive value is not required but recommended in this case, otherwise exposed hosts will not activate (become infectious). If `E_R_rate` is not specified either, then all hosts that become exposed will never change the status, likely leading to undesirable outcomes.

- `E_R_rate`: [*OPTIONAL] (`float`)
  Required if `method` is specified as `SLiM_burnin_epi` and that `-latency_prob` is also specified to be greater than 0. The probability of an exposed host recovering during each tick. The default value is 0, meaning that exposed hosts will never recover.

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

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

- `I_E_rate`: [*OPTIONAL] (`float`)
  Required if `method` is specified as `SLiM_burnin_epi`. The probability of an infectious host deactivating (going to exposed) at each tick. The default value is 0, meaning that infectious hosts will not turn into the exposed state.

- `R_S_rate`: [*OPTIONAL] (`float`)
  Required if `method` is specified as `SLiM_burnin_epi`. The probability of a recovered host losing immunity (going back to susceptible) at each tick. The default value is 0, meaning that infected hosts will not

go back to the susceptible state. A positive value is not required but often recommended to be specified, as initial sequences will only be sampled in last tick of the burn-in, and often the pathogens die out before the last generation/tick.

- `random_seed`: [*OPTIONAL] (`int`)
  Optional parameter to make the result reproducible if desired. Using the same parameter ensures identical outputs when the scripts are rerun.

2.2.1 User input

If the user wants to provide their own initial sequences' mutation information, they needs to make sure the provided file is in VCF format. The `.vcf` file should contain the exact number of samples as what is provided in `-num_seq_init`, and the sample names do not matter, as the initial sequences will be named based on the order in the provided `.vcf` file from 0 to `${NUM_INIT_SEQ}` $-$ 1. If the provided `.vcf` file has more samples than what is specified in `-num_seq_init`, *SeedGenerator* will only take the first `${NUM_INIT_SEQ}` columns as initial sequences' 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`. To run this mode:

```
python seed_generator.py \
    -wkdir ${WKDIR} \
    -num_init_seq ${NUM_INIT_SEQ} \
    -method user_input \
    -path_init_seq_phylogeny ${PATH_TO_INIT_SEQ_PHYLOGENY}
```

Listing 2.10: SeedGenerator Usage user_input

SeedGenerator will create a new folder named `originalvcfs` under the provided working directory, and write one `.vcf` file for each initial sequence in the `${WKDIR}/originalvcfs` directory. They will be named as `seed.X.vcf` where $X \in \{0, \cdots, \$\{NUM\_INIT\_SEQ\} - 1\}$. If `-path_init_seq_phylogeny` is specified, the user needs to make sure that the provided file is in a proper NWK format, where tip labels of the tree should be numbers from 0 to `${NUM_INIT_SEQ}` $-$ 1, corresponding to the initial sequences' 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 check the existence of `seeds.nwk` under working directory. If it is detected, then a complete transmission tree of concatenated initial sequences' phylogeny and outbreak transmission tree will be produced. Please see more details in Chapter 3.

2.2.2 burn-in by Wright-Fisher model

The user can generate initial pathogen genomes with some desired genetic diversity in this mode. This is the option to run a burn-in process using a Wright-Fisher model with SLiM backend. We run the WF model for a specified number of ticks/generations specifed by `-n_gen` in a constant population size specified by `-Ne`. Here is a template for how to initiate the burn-in process:

```
python seed_generator.py \
    -wkdir ${WKDIR} \
    -num_init_seq ${NUM_INIT_SEQ} \
    -method SLiM_burnin_WF \
    -ref_path ${REF_PATH} \
    -mu ${MU} \
    -Ne ${Ne} \
    -n_gen ${NUM_GENS_BURNIN}
```

Listing 2.11: SeedGenerator Usage SLiM_burnin_WF

At the last generation/tick in *SeedGenerator*, initial sequences will be sampled from the current population. *SeedGenerator* creates a new folder named `originalvcfs` under the provided working directory, and write one `.vcf` file for each initial sequence in the `${WKDIR}/originalvcfs` directory, which will be named as `seed.X.vcf` where $X \in \{0, \cdots, \$\{NUM\_INIT\_SEQ\} - 1\}$. *SeedGenerator* records the phylogeny of the sequences sampled, but they do not always coalesce (i.e. can all be traced back to the same ancestor). If they do coalesce, the initial sequences' phylogeny will be written to the working directory, named as `seeds.nwk` in NWK format. If they do not coalesce, i.e., having multiple roots, *SeedGenerator* creates a folder named `seeds_phylogeny_uncoalesced` in the working directory, and write down initial sequences' phylogeny for each root respectively in `.nwk` format in `${WKDIR}/seeds_phylogeny_uncoalesced`.

For example, 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 to the mutation rate of COVID-19 per day in real-time scale. We can run the burn-in for 3650 ticks to simulate 10 years' burn-in period in real-time scale. Let us use an effective population size of 1000 and sample 5 initial sequences:

```
python seed_generator.py \
    -wkdir ${WKDIR} \
    -num_init_seq 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, the user must run *NetworkGenerator* program before they proceeds to this step. It is required 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 generation/tick with no genetic effects. To run this mode:

```
python seed_generator.py \
    -wkdir ${WKDIR} \
    -num_init_seq ${NUM_INIT_SEQ} \
    -method SLiM_burnin_epi \
    -ref_path ${REF_PATH} \
    -mu ${MU} \
    -n_gen ${NUM_GENS_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 recommended for the user to try 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 an absorbing dead end since that would cause an epidemic dying out. If the

number of existing pathogens is less than the desired quantity of initial sequences, the program should 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 sufficient pathogens are present, *SeedGenerator* creates a new folder named `originalvcfs` under the provided working directory, and write one `.vcf` file for each initial sequence in the `${WKDIR}/originalvcfs` directory, which will be named as `seed.X.vcf` where $X \in \{0, \cdots, \texttt{\${NUM\_INIT\_SEQ}} - 1\}$.

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

For example, 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 generations/ticks to simulate 10 years' burn-in period in real-time scale. At this point, we should have already run *NetworkGenerator*, so say we have this contact network with 10000 hosts. We might consider the individual with ID 256 as a cool host to start with. Meanwhile, we want to specify an epidemiological model under which each infected host infects each of its connected host by a probability of 0.03, and all infected hosts will go through a latent stage upon infection. Since the latency period for COVID-19 is usually 6 days after infection, we would have a rate of activation being $0.16(\approx \frac{1}{6})$. Since the recovery period for COVID-19 is usually 28 days after infection, we would have a rate of recovery being $0.035(\approx \frac{1}{28})$. 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. In other words, we would have a rate of immunity loss being $0.018(\approx \frac{1}{56})$. We do not want to include the transitions where infected hosts go back to latent, or latent hosts recover. Finally, we want to choose 5 initial sequences. To realize the specification, we run the following:

```
python seed_generator.py \
    -wkdir ${WKDIR} \
    -num_init_seq 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

For `-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 effects refer to the fitness of pathogens induced by mutations in specific genetic regions in their genomes. EnivolCrossing supports two kinds of fitness operation based on genetic effects, which are transmissibility and drug-resistance. The *GeneticEffectGenerator* program determines the structure of the genetic elements in the pathogen genomes by assigning effect sizes of mutations in those regions. This program can be skipped if users wish to run a simulation without any fitness component. It is mandatory to execute this program if the user wants to 1) provide their own genetic effect model or 2) 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 represent meta information about the regions: names of the regions, starting positions of the regions, and ending positions of the regions. Other columns represent the effect sizes of that region for each trait. Note that traits can be either transmissibility or drug-resistance, specified by column names. Futhermore, we support more than one trait for transmissibility and/or drug-resistance. In the *OutbreakSimulator*, exactly one effect size set has to be chosen for transmissibility and drug-resistance respectively for each epoch.

The program will also calculate the trait values for all initial sequences' mutation profiles based on the assigned effect sizes, thus it is required to run *SeedGenerator* before running *GeneticEffectGenerator*. The calculated trait values will be stored in `${WKDIR}/seeds_trait_values.csv`. Though this file is not a prerequisite for running *OutbreakSimulator*, it might be helpful to check out before executing *HostSeedMatch* to choose appropriate hosts to take specific initial sequences.

A typical command to run *GeneticEffectGenerator* is:

```
python genetic_effect_generator.py \
    -wkdir ${WKDIR} \
    -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 be the same for all programs.
- `method`: [*REQUIRED] (`string`)
  The method of generating effect sizes, 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 relevant parameters.
- `effsize_path`: [*OPTIONAL] (`string`)
  Required if `method` is specified as `user_input`. Absolute path to the effect size file user provided.
- `gff`: [*OPTIONAL] (`string`)
  Required if `method` is specified as `randomly_generate`. Absolute path to the GFF like file that stores the annotation of the genomes.
- `trait_n`: [*OPTIONAL] (`JSON string`)
  Required if `method` is specified as `randomly_generate`. The dictionary specifies the number of traits related to `'transmissibility'` and `'drug_resistance'`.
- `causal_size_each`: [*OPTIONAL] (`list of ints`)
  Required if `method` is specified as `randomly_generate`. It represents 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 floats`)
  Required if `method` is specified as `randomly_generate`. Lower bound on 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 floats`)
  Required if `method` is specified as `randomly_generate`. Upper bound on 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`.

- `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 ints`)
  Required if `method` is specified as `randomly_generate` and `normalize` is specified as `true`. Number of generations/ticks that is expected to be run in *OutbreakSimulator*.

- `mut_rate`: [*OPTIONAL] (`list of floats`)
  Required if `method` is specified as `randomly_generate` and `normalize` is specified as `true`. Mutation rate that is expected to be used in *OutbreakSimulator*.

- `random_seed`: [*OPTIONAL] (`int`)
  Optional parameter to make the result reproducible if desired. Using the same parameter ensures identical outputs when the scripts are rerun.

2.3.1 User input

If the user wants to provide their effect size file, they 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. To run this mode:

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

Listing 2.16: GeneticEffectGenerator Usage user_input

GeneticEffectGenerator will read the user-provided effect size file, check the format and entry of the file, and 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 transmissibility and drug-resistance is activated.

2.3.2 Generate from GFF file

The user can also randomly generate an effect size file with a genome annotation file as an input. A genome annotation file for the reference genome can be downloaded from NCBI. Please make sure that all entries in the `.gff` files do not overlap in terms of genomic positions, or the program might not output correctly. Users can also divide the genome into regions arbitrarily, but it has to be in a `.gff`-like format.

The user should specify a trait set for transmissibility and drug resistance for each epoch. A trait set is a set of effect sizes for the genetic regions selected for such trait. For example, two sets of drug-resistance effect

sizes could be generated, representing resistance for two different types of treatments. In *OutbreakSimulator*, the user could specify two epochs with those two different treatments by activating different sets of effect sizes for drug-resistance.

For each trait set, the number of genetic regions, the lower bound, and the upper bound need to be set. The effect sizes 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 generations/ticks provided. As a result, the expected relative additive average effect size for the pathogen genomes at the end of the simulation will be 1. To run genetic effect generators without normalization:

```
python genetic_effect_generator.py \
    -wkdir ${WKDIR} \
    -method randomly_generate \
    -gff ${GFF_PATH} \
    -trait_n
    '{"transmissibility": ${NUM_SET_TRANSMISSIBILITY}, "drug_resistance": ${
    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

Let us say we want to generate a set of effect sizes for Tuberculosis. Tuberculosis has around 4000 genes, among which we select 5 of them to be causative to higher transmissibility, and we 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 can get very different effect sizes for different genetic regions - we choose between 0.1 and 7. We want to normalize all the effect sizes, since we are running the simulation for 1000 generations/ticks with a mutation rate of 4.4×10^{-8}. This is because we do not 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 ${WKDIR} \
    -method randomly_generate \
    -gff GCF_000195955.2_ASM19595v2_genomic.overlap.gff \
    -trait_n  '{"transmissibility": 1, "drug_resistance": 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. It is recommended to rerun by `user_input` mode after modifications, so that traits values for each seed can be recalculated. This also provides a sanity check on the format. Please note that it is possible that one genetic region has non-zero effect sizes for more than one trait, which provide a framework to simulate **pleiotropy**. This cannot be deliberately specified in random generate mode, but it is 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 on some traits. The user can exploit this possibility to define a gene in which the mutations induce 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 ${WKDIR} \
    -method ${METHOD} \
    -num_init_seq ${NUMBER_OF_INITIAL_SEQUENCES} \
    [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 the same with the directory used for running previous programs. It is assumed that the `contact_network.adjlist` file is 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 initial sequences (`match_scheme`) and corresponding matching parameters (`match_scheme_param`).

- `num_seq_init`: [*REQUIRED] (`int`)
  Total number of initial sequences to be matched to the hosts.

- `path_matching`: [*OPTIONAL] (`string`)
  Required when `method` is set to `user_input`. 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 initial sequence-host to be matched.

- `match_scheme`: [*OPTIONAL] (`JSON string`)
  Required when `method` is set to `randomly_generate`. A dictionary specifying the matching method for each initial sequence. The values should be one of `["ranking", "percentile", "random"]`. If all the initial sequences are to be matched randomly, it can be "`random`".

- `match_scheme_param`: [*OPTIONAL] (`JSON string`)
  Required when `method` is set to `randomly_generate`. A dictionary containing parameters for each matching method. Refer to the sections below for the format of parameters corresponding to each matching method.

- `random_seed`: [*OPTIONAL] (`int`)
  Optional parameter to make the result reproducible if desired. Using the same parameter ensures identical outputs when the scripts are rerun.

2.4.1 User-provided matching file input

Users have the option to match the initial sequences 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 ${WKDIR} \
        -method user_input \
        -num_init_seq 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 initial sequences 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 ${WKDIR} \
    -num_init_seq ${NUMBER_OF_INITIAL_SEQUENCES} \
    -method randomly_generate
```

Listing 2.21: HostSeedMatch random matching example

Alternatively, the program allows users to match the initial sequences to hosts based on user-specified methods and parameters. For the understanding of the following methods, we would like to mention that hosts are sorted in reverse order based on their number of contacts in the host contact network.

Ranking Method (`'ranking'`)

- Requires an integer to specify the rank of a desired host.
- Hosts with higher connectivities 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 higher connectivities have lower percentiles .

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 initial sequences. For the first initial sequence, we just want to randomly assign a host. However, for the second initial sequence, it is desired to match it with a host possessing the highest number of contacts within the network. The third initial sequence 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 ${WKDIR} \
    -num_init_seq 3 \
    -method randomly_generate \
    -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

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** (e.g., `${WKDIR}`) 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 the user would like to incorporate any genetic effects to fitness in their model. Generated by *GeneticEffectGenerator* (Section 2.3)
- `originalvcfs` (and the `seed.X.vcf` inside): Required if the user would like to use initial sequences different from the reference genome. Generated by *SeedGenerator* (Section 2.2)
- `seeds.nwk`: Optional. Generated by *SeedGenerator* (Section 2.2). If the file exists, *OutbreakSimulator* can generate the full phylogeny by binding the transmission tree to the initial sequences' 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. To execute `OutbreakSimulator`, users can just run:

```
1  python outbreak_simulator.py \
2      -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), we will introduce the modules of `OutbreakSimulator` and how the simulation actually works. In Chapter 5, we will walk you through some examples about how to simulate your own model.

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 (full config). The full config file is needed if you want to run streamlined preprocessing and outbreak simulation (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`)
  The number of replicates to run with the same set of configurations (specified by this file).
  (`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`)
  The number of generations/ticks the user wants 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 for 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 available: `"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 available: `"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 during each tick. This parameter will not be applied unless `"within_host_reproduction"` is set to `true`.
- `"cap_withinhost"`: (`int`)
  The maximum number of pathogens that is allowed to exist in one host. i.e., if the cap is reached, then no within-host reproduction or super-infection could happen for this host. The parameter should be set to 1 if no within-host reproduction or super-infection is activated.

3.2.3 SeedsConfiguration

This section specifies the initial sequences.

- `"seed_size"`: (`int`)
  Number of initial sequencess. Should be the same as what is used for *SeedGenerator* and *HostSeedMatcher* unless `"use_reference"` is set to `true`, in which case should be the same as what is used for *HostSeedMatcher*.

- `"use_reference"`: (`binary`) `true` or `false`
  Whether to use the reference genome as the initial sequence. If `true`, then it is not required to run *SeedGenerator* in the preprocessing pipeline.

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 `true`, then it is required to run *GeneticEffectGenerator* in the preprocessing steps.

- `"ref_path"`: (`string`)
  Absolute path to the reference genome in `.fasta` format. It should be the same as the one used for *SeedGenerator* if applicable.

- `"traits_num"`: (`JSON string`)
  The dictionary specifies the number of traits/effect size sets related to `'transmissibility'` and `'drug_resistance'`. e.g., `"traits_num": '{"transmissibility": 1, "drug_resistance": 2}'` represents 1 set for transmissibility and 2 sets for drug-resistance. This parameter 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 underlying transmission network. If the parameter is set to `false`, then it is not required to run *NetworkGenerator*, and a random branching process will be run instead (STILL UNDER DEVELOPMENT).

- `"host_size"`: (`int`)
  host population size. It should be set to 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.

- `"slim_replicate_seed_file_path"`: (`string`)
  Absolute path to a `.csv` file with a single column `random_number_seed` that stores the random seed to use for the reproducibility of results for the replicates. If left empty, the program will run but does not guarantee reproducibility across runs of the simulations.

- `"model"`: (`string`)
  The compartmental model that will be applied. The user can 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 ints`)
    The generations/tick(s) at which the simulation should switch to the next set of parameters. 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 ints`) 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

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

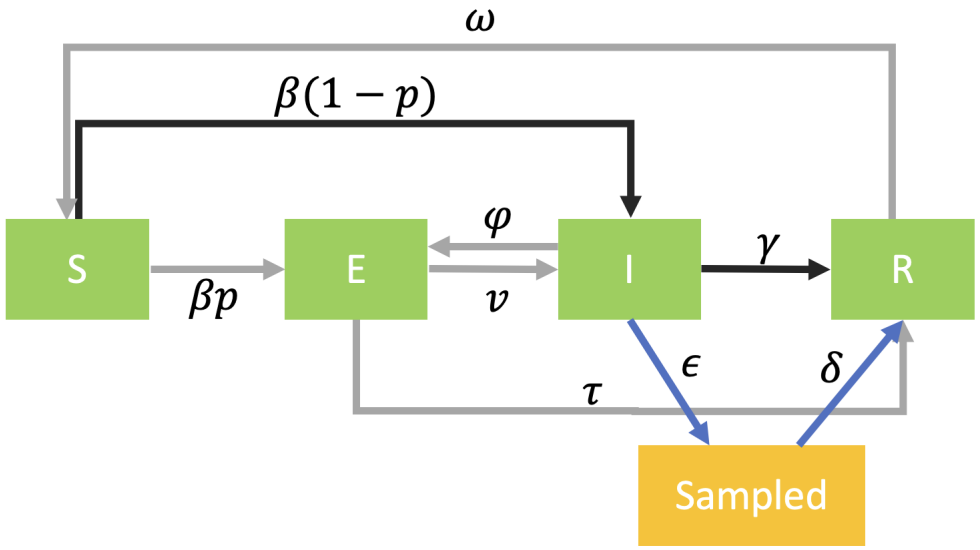
$$\text{Trait}_i = \Sigma_{j=0}^{n} q_j \times x_{ij}$$

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

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 ${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 -u seed_generator.py \
    -wkdir ${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 ${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 ${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 -u outbreak_simulator.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).

6. Now you should be able to view outputs in each replicate's subfolder and in the working directory. For detailed instructions on how to read the outputs, please refer to Chapter 6.

5.2 Multi-stage drug treatment

In this section, we will simulate a COVID-19 outbreak in a host population of size 10,000 for 2 years. The time scale is that we want each tick to represent 1 day, so we want to simulate 730 ticks. The mutation rate of coronavirus is 6.67×10^{-3} SNPs/bp/year, which is 1.8×10^{-5} SNPs/bp/day. In this simulation, we are going to apply realistic host population structure and different treatment stage.

1. The first step is to run *NetworkGenerator*. We want to simulate a realistic host population structure using a scale-free graph. We thus run

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

```

Listing 5.7: NetworkGenerator model 2

2. We then run *SeedGenerator*. In this example, we are starting from the reference genome, thus this step could be omitted.

3. We then run *GeneticEffectGenerator*. In this example, we are using a pre-defined effect size file (the provided effect size file).

```
    python -u genetic_effect_generator.py \
        -wkdir ${WKDIR} \
        -method user_input \
        -effsize_path ${EFFSIZE_PATH}$ \
        -trait_n '{"transmissibility": 1, "drug_resistance": 2}' \
```

```
        -n_seed 1

```

Listing 5.8: GeneticEffectGenerator model 2

Here `-trait_n 1 2` tells EnivolCrossing that in the provided effect size file, there are 3 trait sets, of which the first one is transisibility and the second 2 is for drug resistance. You should be able to see a `causal_gene_info.csv` appearing in your working directory, along with a `seeds_trait_values.csv` file that only has a header, since we don't have any seed sequence available now.

4. We then run *HostSeedMatcher*. In this example, we are using only one seed, and we want to match it to a host that has median contact degree. We thus utilize the ranking method to match seed and host.

```
    python -u seed_host_matcher.py \
        -wkdir ${WKDIR} \
        -method randomly_generate \
        -n_seed 1 \
        -match_scheme '{"0": "ranking"}' \
        -match_scheme_param '{"0": 5000}'

```

Listing 5.9: SeedHostMatcher model 2

You should see a `seed_host_match.csv` appearing in your 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 days, recovery time can be 2 weeks. We want to model that all new infections lead to latency at first. Though it's controversial, but immunity can lose in 30 days. For basic transmission rate, we assume an arbitrary probability of 0.03, meaning that for each contact pair, an everyday infection probability is 3%. 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_drugresist",
    "n_replicates": 3
  },
  "EvolutionModel": {
    "n_generation": 730,
    "mut_rate": 1.8e-6,
    "trans_type": "additive",
    "dr_type": "additive",
    "within_host_reproduction": false,
    "within_host_reproduction_rate": 0,
    "cap_withinhost": 1
  },
  "SeedsConfiguration": {
    "seed_size": 1,
    "use_reference": true
  },
  "GenomeElement": {
    "use_genetic_model": true,
    "ref_path": "/Users/px54/Documents/TB_software/V2_code/test/data/COVID/
    EPI_ISL_402124.fasta",
    "traits_num": {"transmissibility": 1, "drug_resistance": 2}
  },
  "NetworkModelParameters": {
    "use_network_model": true,
```

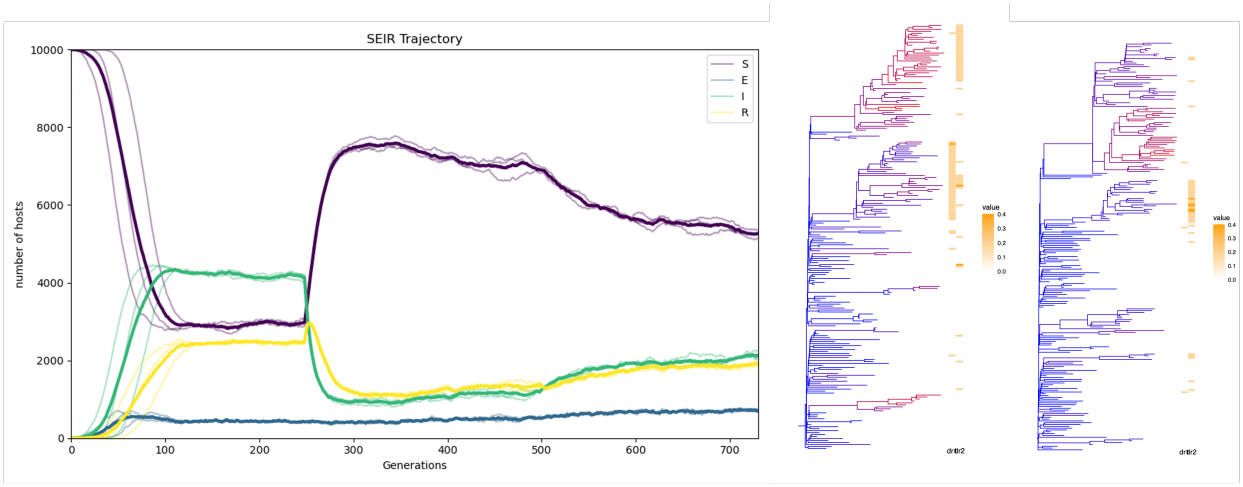
```
    "host_size": 10000
  },
  "EpidemiologyModel": {
    "model": "SEIR",
    "epoch_changing": {
      "n_epoch": 3,
      "epoch_changing_generation": [250, 500]
    },
    "genetic_architecture": {
      "transmissibility": [1, 1, 1],
      "cap_transmissibility": [10, 10, 10],
      "drug_resistance": [0, 1, 2],
      "cap_drugresist": [0, 10, 10]
    },
    "transiton_rate": {
      "S_IE_rate": [0.04, 0.04, 0.04],
      "I_R_rate": [0.03, 0.06, 0.05],
      "R_S_rate": [0.05, 0.1, 0.1],
      "latency_prob": [1, 1, 1],
      "E_I_rate": [0.3, 0.3, 0.3],
      "I_E_rate": [0, 0, 0],
      "E_R_rate": [0, 0, 0],
      "sample_rate": [0.0001, 0.0001, 0.0001],
      "recovery_prob_after_sampling": [0, 0, 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 5.10: config model 2

You should then see the simulator running, the steps it's currently running will be printed out in the standard output. Since this is a stochastic simulation, it's not guaranteed that everytime the outbreak can become an endemic stage, i.e. it might end very early that you do not get a lot of samples. If no samples exist, there will be no post-simulation processing steps, and you will receive a warning for that.

6. Now you should be able to view outputs in each replicate's subfolder and in the working directory. For detailed instructions on how to read the outputs, please refer to Chapter 6. Here we are showing example outputs for this special case. We are showing one aggregated SEIR trajectory for three replicates, where you can clearly see the different behaviors of the trajectory in different epochs. We are also showing the examples of the transmission trees for two replicates, where you can clearly see that the sublineages that acquire drug-resistance get to transmit more after the drug-treatment epochs get activated. They also get to accumulate transmissibility-conferring mutations over time and becomes

overall more transmissibible as time proceeds. Also, as the 2nd drug-resistance epoch get activated, sub-lineages that has drug-resistance mutations for the 2nd epoch's drug get to transmit more as well.



Part III

Process the output data

Chapter 6

Output

Contents

In this chapter, we are going to talk about the structure of the output of EnivolCrossing and introduce how to read the outputs.

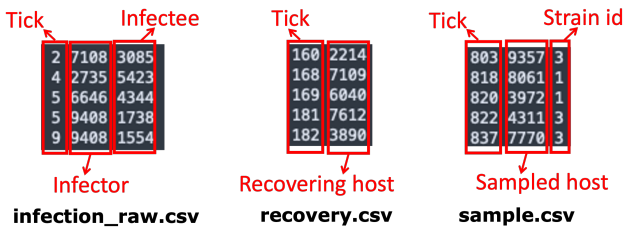
6.1 Output structure

For every replicate, Enivolcrossing creates a folder named by the replicate id (starting from 0) in the working directory. In each replicate folder, there will be the following outputs:

- `infection_raw.csz.gz`: Compressed csv file that stores all the raw infection events that ever happened. Every row in the file represents one infection event. The first column represents tick, the second column representing infector's host id, and the third represents the infectee's host id.

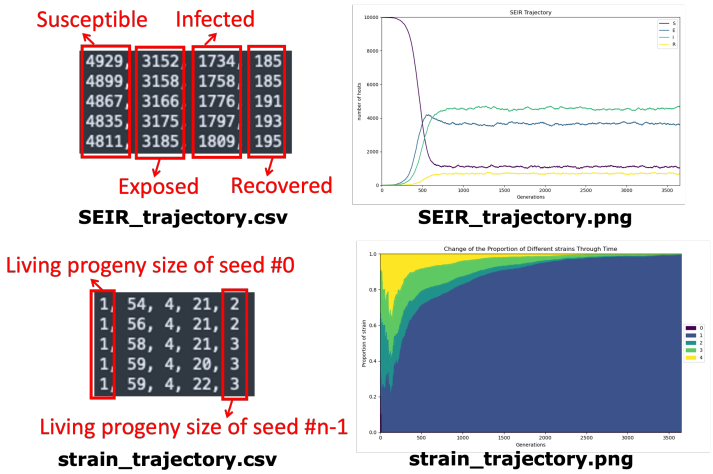
- `recovery.csz.gz`: Compressed csv file that stores all the recovery events that ever happened. Every row in the file represents one recovery event. The first column represents tick, the second column representing the recovering host id.

- `sample.csz.gz`: Compressed csv file that stores all the sampling events that ever happened. Every row in the file represents one recovery event. The first column represents tick, the second column representing the sampled host id.



- `SEIR_trajectory.csv.gz`: Compressed csv file that stores the SEIR compartmental size for each tick. It has the row number equals to the total number of ticks. The four columns represents the size of susceptible, exposed, infected and recovered respectively.

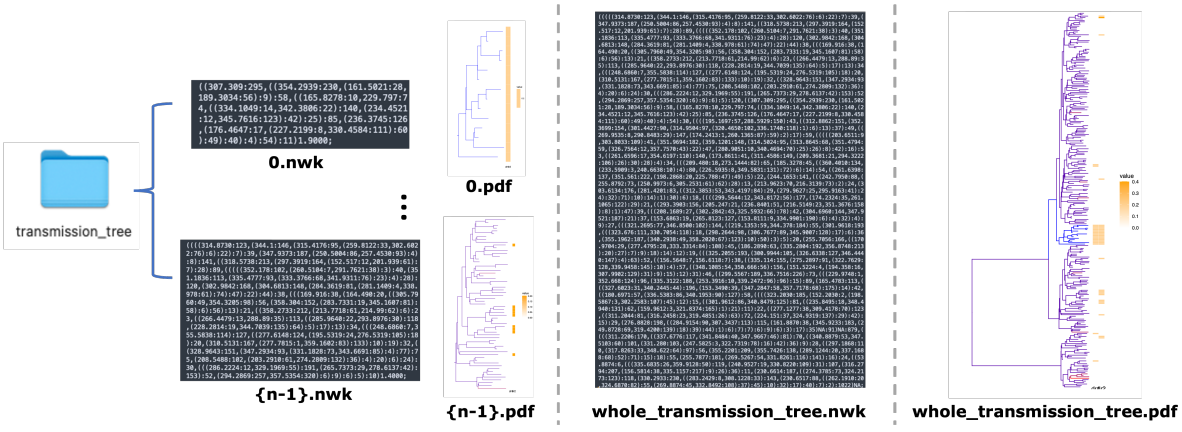
- `strain_trajectory.csv.gz`: Compressed csv file that stores the size of the progeny of each strain for each tick. It has the row number equals to the total number of ticks. Each column represents size of the progeny of one seed (strain), in the order of seeds' ids.
- `SEIR_trajectory.png`: Plotted `SEIR_trajectory.csv.gz`.
- `strain_trajectory.png`: Plotted `strain_trajectory.csv.gz` in proportion.



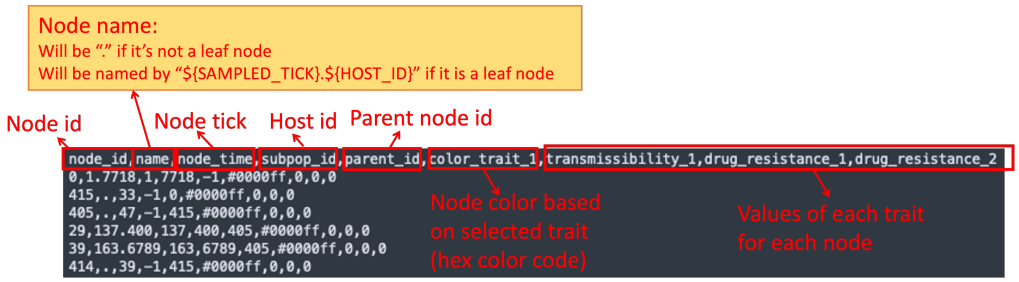
- `transmission_tree`: A folder that stores transmission tree for each seeds' progeny, within which `x.nwk` for each seed id x will present. For seeds that has more than one sampled offspring, a `tree.x.pdf` that is a plotted tree will be present.

  The leaf labels in the `x.nwk` file is in the format of `${SAMPLED_TICK}.${SAMPLED_HOST_ID}`, which corresponds to the sample names in the `.vcf file`

- `whole_transmission_tree.pdf`: Plotted transmission tree, concatenating the seeds' phylogeny and each seed's transmission tree. Presents only one seed phylogeny presents in the working directory.
- `whole_transmission_tree.nwk`: Transmission tree, concatenating the seeds' phylogeny and each seed's transmission tree. Presents only one seed phylogeny presents in the working directory.



- `sampled_genomes.trees`: Treesequence of the sampled genomes, output by SLiM in `OutbreakSimulator`.
- `transmission_tree_metadata.pdf`: Metadata of the transmission tree, storing the trait values for each node in the transmission tree and the coloring of each node.

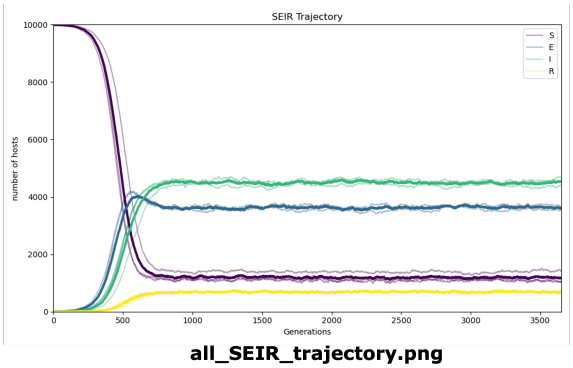


transmission_tree_metadata.nwk

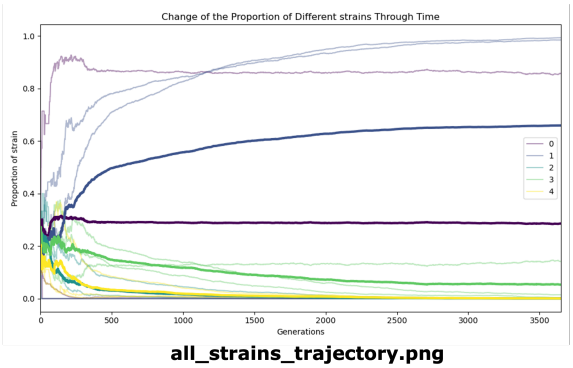
- `sampled_pathogen_sequences.vcf`: The mutation profiles of the sampled pathogen sequences in `vcf` format, where sample names correspond to the naming criteria in `whole_transmission_tree.nwk`.

In the working directory, there are going to be some outputs as well:

- `all_SEIR_trajectory.png`: Plot the aggregated results from `SEIR_trajectory.csv.gz` for each replicate, with the average trajectory plotted out.
- `all_strain_trajectory.png`: Plot the aggregated results from `strain_trajectory.csv.gz` for each replicate, with the average trajectory plotted out.



all_SEIR_trajectory.png



all_strains_trajectory.png

6.2 Specifying tree plotting options

In the config file for running the simulation, there is a section called `"Postprocessing_options"`, which specifies the plotting options of the tree.

- `"do_processing"`: (`binary`) `true` or `false`. Whether to do post-simulation data-processing. If `false`, then treesequence will not be processed, thus plotting of the transmission trees and the nwk file for the whole transmission tree will not be present in the working directory.
- `"tree_plotting"`: Parameters for the tree plotting, used if `"do_processing": true`.
  - `"branch_color_trait"` (`integer`) Which trait value will be used for branch color when plotting the tree. By default it is going to be 1, the first trait set. Note that here drug-resistance traits and transmissibility traits are counted together, for example if you have 1 transmissibility trait and 2 drug-resistance traits, that's 3 traits in total, and you want to color by the first drug-resistance trait, you should enter 2 here (1 refers to the transmissibility trait). You can also specify 0, which means color by seed id. The tree will be colored based on which seed does the node originate from, and the color will be the same set as what is in `strain_trajectory.png`. Note that if all nodes and samples has the trait value being 0 for the trait you select, the tree will be colored in black.

- `"drug_resistance_heatmap"` (`binary`) Whether to plot the headmap for drug-resistance. If `false`, then the tree plotted will not include the heatmap representing drug-resistance.

Part IV

Streamline

Part V

GUI