Bridging forward-in-time and coalescent simulations using pyslim

Shyamalika Gopalan[1], Murillo F. Rodrigues[2,3], and Peter L. Ralph[3,4]

[1]Department of Genetics and Biochemistry and Center for Human Genetics, Clemson University
[2]Division of Genetics, Oregon National Primate Center, Oregon Health & Science University
[3]Department of Biology and Institute of Ecology and Evolution, University of Oregon
[4]Department of Mathematics, University of Oregon

May 1, 2025

Abstract

Lorem ipsum

Introduction

Simulations have been an invaluable tool in population genetics for the past six decades. The two main strategies for population genetic simulation differ in the direction of the process: forward or backward-in-time. The coalescent process models the ancestry of sampled genomes back in time until they coalesce into one most common recent ancestor (MRCA). It is perhaps the most common framework for population genetic simulation because of its efficiency: it bypasses the need to represent entire populations in memory and the sampling of gametes every generation. Despite its efficiency, the coalescent has strict assumptions (e.g., neutrality) which limits applicability. Forward-in-time simulations starts with actual individuals and applies evolutionary rules (e.g., mutation, recombination, selection) over generations until a certain criterion is met. Thus, forward-in-time simulations are much more flexible, but they come with a high computational cost.

Recent advancements both in computational power and software development have made simulations much more accessible and popular. A key development that has decreased the computational cost of simulations is the tree sequence, a data structure that concisely encodes correlated genealogies along the genome. In the context of forward-in-time simulations, the recording of tree sequences increases efficiency because

it allows for (i) the omission of neutral mutations during the simulation process, and (ii) the use of fast coalescent as a neutral "burn-in" phase, such that the forward simulation can begin with an equilibrium level of genetic diversity. Further, using tree sequences it is possible to bridge forward and backwards methods, for example by using the coalescent to ensure all lineages coalesce into a single common ancestor (a process called r̈ecapitation̈).

Here, we present `pyslim`, a Python package for reading and modifying `tskit` tree sequences produced by the popular coalescent and forward-in-time simulation tools such as `SLiM` and `msprime`. `pyslim` provides a way to perform ḧybrid s̈imulations, combining both forwards and backwards (coalescent) methods. We will describe the main uses of `pyslim`: (i) recapitation, which is the process of filling in the history of the first-generation individuals which have not coalesced, (ii) generation of initial diversity for forward-in-time simulations, (iii) parallelization of multi-species simulations, and (iv) complex simulation of alternating life cycles.

Starting with diversity generated by coalescent simulation

Simulations of large populations with selection can be costly, especially because we might need to run a lengthy "burn-in" period to get the genetic variation for selection to act on. Because the precise form of the burn-in may not be important, a neutral burn-in can be efficiently run with coalescent simulator such as `msprime`. In this section, we will demonstrate how to perform such a hybrid simulation using `pyslim`, `msprime` and `SLiM`.

Imagine the scenario where we perform a lab experiment in which we take high-diversity organisms from the wild and subject them to selection for a few dozen generations. Although genetic diversity in the wild is most likely not neutral, we do not know precisely what it does look like and a coalescent simulation would be an acceptable starting point. The key attribute of reality we would like to approximate is the joint distribution of allele frequencies and effect sizes. If the alleles affect a trait under stabilizing selection, we would expect a negative correlation between the two. On the other hand, there would be no relationship between allele frequencies and effect sizes if the trait we are selecting on in the lab is not under strong selection in the wild. We will simulate the scenario where the trait we put under selection in the lab is not under stabilizing selection in the wild. To do so, we will:

1. Run a coalescent simulation with msprime.
2. Add `SLiM` metadata to the nodes, individuals, and populations.
3. Add `SLiM` mutations with msprime, and edit the mutation metadata to assign selection coefficients.
4. Run the `SLiM` portion of the simulation.
5. Do some descriptive analysis of the results of selection.
6. Add neutral mutations to the tree sequence.
7. Do some descriptive analysis of genetic diversity along the genome.

Parallelizing forward-in-time simulations of multiple species

Any two branches stemming from the same node in a species tree are independent from each other and thus can be simulated in parallel (assuming no migration between the species). For example, in the phylogeny depicted in Figure 1, branches of the same color can be simulated in parallel. To do so, we will need to (i) simulate the history of each branch and (ii) join the resulting simulations together onto one multi-species history.

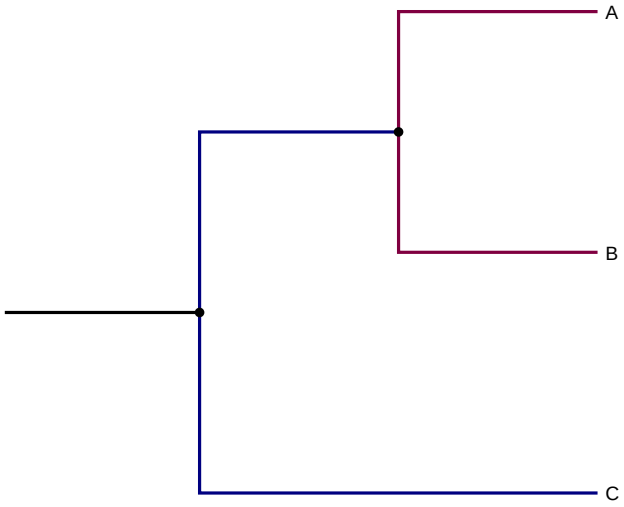

Figure 1: Example of phylogeny we might want to simulate. Note how branches with the same color can be simulated in parallel when there is no migration.

0.1 Parallel simulation of branches

First, we need to write a `SLiM` script that will be used for simulating the history of each branch in our phylogeny. We will perform a simple simulation, in which each branch can have a different (but fixed) population size and length (number of ticks). Also, we will allow deleterious mutations to happen across the entire chromosome at a fixed rate. See the code below (Listing 1).

```
// The following constants need to be defined:
// - outfile: path to save the tree sequence of the simulation.
// - popsize: population size.
// - num_gens: run simulation for num_gens ticks.
// - infile: a string with path to tree sequence to start from; optional.
initialize()
{
    initializeSLiMModelType("WF");
    initializeTreeSeq();
    initializeMutationRate(1e-8);
    initializeMutationType("m1", 0.5, "f", -0.01);
    initializeGenomicElementType("g1", m1, 0.1);
    initializeGenomicElement(g1, 0, 1e6-1);
    initializeRecombinationRate(1e-9);
}

1 late() {
    // if no input tree sequence is provided, then start a subpopulation
    if (infile == "") {
        p = sim.addSubpop("p1", popsize);
    } else {
        // reloading must happen in late()
        sim.readFromPopulationFile(infile);
        parent = sim.subpopulations[0];
        p = sim.addSubpopSplit(max(sim.subpopulations.id) + 1, popsize, parent);
        parent.setSubpopulationSize(0);
    }
    p.name = popname;
}

// schedule the end of the simulation
1 late() {
    finaltick = num_gens + community.tick;
    community.rescheduleScriptBlock(s0, ticks=finaltick);
}

// event that saves the tree sequence
s0 1000 late() {
    sim.treeSeqRememberIndividuals(sim.subpopulations.individuals);
    sim.treeSeqOutput(outfile);
}
```

Listing 1: Simple `SLiM` script to simulate a constant size population that can be started from an existing tree sequence.

For each branch, the presence or absence of `infile` tells `SLiM` whether a previous branch exists or not. If so, `SLiM` will read the previous tree sequence and change the

population size accordingly. Note that when you read a tree sequence into `SLiM`, the tick counter will be updated with the time encoded in the tree sequence, so we need to set the end of the simulation as the length of the branch (`num_gens`) plus the current "time" at the end of the loaded tree sequence. At the end of the simulation, we call `sim.treeSeqRememberIndividuals` right before saving the resulting tree sequence. This is necessary because we need to ensure the individuals in the final generation are never dropped from the tree sequence in future runs of `SLiM` which are started from the output of the simulation, as they will later be used to glue the tree sequences together.

The example phylogeny we will simulate is encoded in the table below (Table 1).

Child	**Parent**	**Population size**	**Edge length**
root	-	500	2000
C	root	50	250
I	root	100	200
B	I	70	50
A	I	40	50

Table 1: Parameters of the phylogeny that will be simulated.