[Journal name]
[Article]

Title:

Practical Applications of Tree-Sequence Recording using SLiM 3 and msprime

Authors:

Benjamin C. Haller ‡
Dept. of Biological Statistics and Computational Biology
Cornell University, Ithaca, NY 14853, USA

Jared Galloway
Institute of Ecology and Evolution
University of Oregon, Eugene, OR 97403, USA

Jerome Kelleher
Wellcome Trust Centre for Human Genetics
University of Oxford, Oxford OX3 7BN, United Kingdom

Philipp W. Messer *
Dept. of Biological Statistics and Computational Biology
Cornell University, Ithaca, NY 14853, USA

Peter L. Ralph ‡ *
Institute of Ecology and Evolution
University of Oregon, Eugene, OR 97403, USA

‡ Corresponding authors
* Joint senior authors

Corresponding Authors:

Benjamin C. Haller, bhaller@benhaller.com

Peter Ralph, plr@uoregon.edu

Author Contributions (alphabetical by last initial):

Wrote the tree-sequence recording code in SLiM 3 and/or msprime: JG BCH JK PLR
Designed and executed the examples: BCH PLR
Wrote the paper: BCH
Senior authors providing support: PWM PLR

Abstract

Forward simulations are increasingly important in population genetics and evolutionary biology, and there is an increasing desire to run large simulations involving full genome sequences, large populations, complex demography, ecological influences, explicit space, spatial interactions, and other types of realistic dynamics. However, such models can take an extremely long time to run – often so long that it is not possible to model the things we would like to model. This situation presents an obstacle to the field, and although increases in computing performance are gradually increasing our capabilities, there is a need for more rapid progress. The tree-sequence recording method of Kelleher et al. (2018) provides a path forward. This method has several benefits: (1) it allows neutral mutations to be omitted from forward-time simulations and added later, much more efficiently; (2) it allows rapid computation of some key population genetics statistics along the chromosome; and (3) it provides a compact representation of a population's ancestry that can be analyzed in Python using the msprime package. We have extended the method of Kelleher et al. (2018) to allow the recording of non-neutral mutations in the tree sequence during forward-time simulation in SLiM 3, a free, open-source evolutionary simulation software package, greatly broadening the utility of their method. We here present several examples of the practical application of this method, which we believe opens up whole new horizons of possibility in the modeling of evolutionary processes.

Keywords

[my suggestions, based on things that aren't in our title or abstract but that are relevant…]

pedigree recording, coalescent, background selection, true local ancestry, selective sweeps

Introduction

Forward simulations are increasingly important in population genetics and evolutionary biology. For example, they can be useful for modeling the future of real-world systems (REFS; cite Matz's corals paper here), for discovering the ecological and evolutionary mechanisms that led to present-day genomic patterns in a species (REFS; cite a human-evo paper), and for exploring theoretical ideas about evolution (REFS), among other uses. Because of this broad utility, there is an increasing desire to run simulations with increased realism in a variety of areas: longer simulated loci up to and including full genome sequences, large populations, complex demography, ecological interactions involving other organisms and the environment, explicit space include continuous landscapes with spatial variation in environmental variables, spatial interactions between organisms such as spatial competition and mate choice, and so forth.

However, this type of realism typically comes at a price, in both processing time and memory usage. Since processing time and memory are finite resources, this can make it difficult or, in practical terms, impossible to run some models. Advances in computing power have gradually extended the boundaries of what is possible (REF?), as have performance improvements in some forward simulators (REF cite SLiM 2 paper, fwdpp paper), but these limitations continue to hold back progress in the field, limiting the level of realism that can be attained in models and slowing down the discovery of new results.

From this perspective, the new "pedigree recording" or "tree-sequence recording" method of Kelleher et al. (2018) holds great promise, for several reasons. One reason is that it allows neutral mutations to be omitted entirely from most forward simulations; with the recorded tree-sequence information neutral mutations be accurately overlaid after forward simulation has completed, at a small fraction of the computational cost. The efficiency benefit here comes from the fact that neutral mutations only need to be added along those branches of the ancestry tree that are represented in the final simulation generation; all of the other ancestral branches, which typically comprise the vast majority of the full evolutionary tree, can be ignored since they do not contribute to or influence the ultimate descendants. Since many forward simulations spend the large majority of their time managing neutral mutations, this can improve performance by an order of magnitude or more (Kelleher et al., 2018).

Another reason is that the recorded tree sequences can allow rapid calculation of various statistics along the genome (Kelleher et al., 2018). Some of these statistics are related to the structure of the ancestry tree itself, such as the diversity or true local ancestry at loci along a chromosome. Other statistics are calculated from the mutations that exist in the vicinity of a locus, but in such a manner that they can be calculated much more rapidly if information is re-used from position n to position n+1 in a manner that the tree sequences can facilitate. Calculation of statistics of this sort from simulated data can be very time-consuming, especially when long genomes are involved and many replicate simulation runs have been performed, so the ability to speed up such calculations is important.

A third reason is that the recorded tree sequences themselves provide a record of the pattern of ancestry and inheritance in a population that can be useful. These tree sequences can be loaded into Python, and can then be analyzed and even manipulated in Python to all sorts of purposes. This open-ended flexibility, enabled by the free, open-source msprime package, means that one is not limited to a set of pre-packages ways of using tree sequences. Instead, new ways of using them to improve and accelerate simulations and statistics computation can itself be an area of active research.

Given these advantages, we have worked to integrate tree-sequence recording into the SLiM 3, a new major release of the free, open-source SLiM simulation software package (REF to the SLiM 3 paper if possible, or to the SLiM 2 paper). It is now possible to turn on tree-sequence recording in any SLiM model, and then to output the recorded tree sequences at any point in the simulation. In addition, we have extended the original tree-sequence recording method described by Kelleher et al. (2018) to allow the recording of mutations during forward simulation; these recorded mutations are then output as part of the overall recording. This allows the tree-sequence output format, a `.trees` file, to be used as a way of saving and then restoring the state of a simulation. It also makes it possible to access the mutations that occurred during a forward simulation later in Python, allowing analyses to make use of them.

To illustrate the large advantages provided by tree-sequence recording, and to show how to take advantage of those benefits when using SLiM for forward simulation, we will here present four examples of the practical application of the tree-sequence recording methodology. In the first example, we will show the large performance benefit of tree-sequence recording compared to a traditional forward simulation. The second example will use tree-sequence recording to

efficiently simulate background selection near genes undergoing deleterious mutation. Our third example will be of a model of admixture between two subpopulations, showing how to use the recorded tree sequences to calculate the mean true local ancestry at every position along the chromosome. Finally, the fourth example illustrates how msprime's ability to generated simulated tree sequences using the coalescent can provide a "neutral burn-in" starting point for a SLiM simulation that then models the advent of non-neutral dynamics. These four examples illustrate just four of the many ways in which we believe tree-sequence recording will open up new horizons of possibility for evolutionary modelling.

Examples

All examples were executed on a MacBook Pro (2.9 GHz Intel Core i7, 16 GB RAM) running macOS 10.13.4, using SLiM 3.0, msprime 0.5.1.dev57+g30c2fd1, Python 3.4.8, and R 3.4.3. All reported times were measured with the Un*x tool `/usr/bin/time` (summing the reported user time and system time) at the command line. Peak memory usage for SLiM runs was assessed with SLiM's `-m` command-line option.

Example I: A simple neutral model

Our first example is a model of neutral evolution on a chromosome of length L=1e8 base positions, with mutation rate μ=1e-7 and recombination rate r=1e-8 (both expressed as the event probability per base per generation), in a population of size N=500, running for a duration of G=10N=5000 non-overlapping generations, under panmixis. The SLiM model for this is very simple (code I.1):

```
initialize() {
  initializeMutationRate(1e-7);
  initializeMutationType("m1", 0.5, "f", 0.0);
  initializeGenomicElementType("g1", m1, 1.0);
  initializeGenomicElement(g1, 0, 1e8-1);
  initializeRecombinationRate(1e-8);
}
1 {
  sim.addSubpop("p1", 500);
}
5000 late() {
  sim.outputFull("./ex1_noTS.slimout", binary=T);
}
```

This sets up a single "genomic element" spanning the full length of the chromosome, with neutral mutations of type `m1` generated at the desired rate, and with the desired recombination rate. In generation 1 a new subpopulation of the desired size is created, and the model runs to generation 5000, after which it outputs the full simulation state. The SLiM manual provides additional explanation of these concepts (Haller and Messer, 2016). This model took 212.3 seconds to run, and reached a peak memory usage of 390.8 MB.

This model is easily recast to use tree-sequence recording (code I.2):

```
initialize() {
  initializeTreeSeq();
  initializeMutationRate(0);
```

```
  initializeMutationType("m1", 0.5, "f", 0.0);
  initializeGenomicElementType("g1", m1, 1.0);
  initializeGenomicElement(g1, 0, 1e8-1);
  initializeRecombinationRate(1e-8);
}
1 {
  sim.addSubpop("p1", 500);
}
5000 late() {
  sim.treeSeqOutput("./ex1_TS.trees");
}
```

Here the call to `initializeTreeSeq()` enables tree-sequence recording in SLiM. The mutation rate can then be set to zero; SLiM no longer needs to model the neutral mutations because they will be overlaid in a later step. Finally, a `.trees` file is output at the end of the run, instead of calling SLiM's `outputFull()` method, so that the recorded tree sequence is preserved. In all other respects these models are identical. This is typical of adapting a SLiM model to use tree-sequence recording: in general, the aim is to remove the modeling of neutral mutations while preserving other aspects of the model verbatim.

After that model completes, a simple Python script is executed:

```
python msp-add-mutation.py --tree_file ex1_TS.trees --mut_rate 1e-7
```

This script (provided as Supplemental S1) uses the msprime Python package to overlay neutral mutations upon the recorded tree sequence. This produces exactly the same result, in a statistical sense, as model I.1; it is as if the neutral mutations were included in the forward simulation, except that the vast majority of the bookkeeping work is avoided because mutations only need to be overlaid upon the ancestral genomic regions that persisted to the end of the simulation; mutations arising on branches that did not survive need not be modeled.

The total time to execute the second pipeline is 4.99 seconds (2.35% of the time for script I.1), most of which (4.12 seconds) is spent in SLiM. The peak memory usage during the SLiM run is 134.6 MB (34.5% of the usage for script I.1). Note that the speedup produced by the tree-sequence recording method will vary enormously depending upon the details of the simulation; all of the work to track neutral mutations is eliminated, but new work is added involving the recording of all the recombination events that go into producing the tree sequence. In general, the largest speedup will be observed with very long chromosomes with many neutral mutations; indeed, when modeling a short chromosome the tree-sequence recording overhead can outweigh the neutral-mutation savings, so it is worthwhile to check the runtimes of both methods to find the method with the best performance for a particular model.

This first example can, of course, also be done with a pure coalescent simulation using msprime in Python (requesting $2N$=1000 samples since we are simulating diploids):

```
import msprime
ts = msprime.simulate(sample_size=1000, Ne=1000, length=1e8,
  recombination_rate=1e-8, mutation_rate=1e-7)
ts.dump("ex1_msprime.trees")
```

This is faster still, at 1.413 seconds; the coalescent, when available, will generally be the fastest method. However, the coalescent can only be used in a few simple scenarios such as this; the rest of the examples we present here would not be possible to simulate with the coalescent. Furthermore, whereas conventional forward simulation is ~150 times slower than the coalescent, the time for the forward simulation with tree-sequence recording is only ~3.5 times longer than the coalescent; with such a small difference, it might often make sense to use forward simulation with tree-sequence recording for other reasons, such as workflow pipeline consistency. It is also worth noting that the coalescent is only an approximation of the Wright-Fisher model in diploids, and will diverge from it under certain conditions (REF) – one such condition being a sample size that is large relative to the population size, as is the case here.

Example II: Background selection

Our second example is of background selection, the selection against neutral sites linked to nearby deleterious mutations. Because mutations in coding regions are so often deleterious, background selection has been observed to produce a "dip in diversity" in non-coding regions near genes, with a signature of decreasing genetic diversity with decreasing distance to the nearest gene (REF). Here is a SLiM model that uses tree-sequence recording to model this scenario:

```
initialize() {
  defineConstant("N", 10000);  // pop size
  defineConstant("L", 1e8);    // total chromosome length
  defineConstant("L0", 200e3); // between genes
  defineConstant("L1", 1e3);   // gene length
  initializeTreeSeq();
  initializeMutationRate(1e-7);
  initializeRecombinationRate(1e-8, L-1);
  initializeMutationType("m2", 0.5, "g", -(5/N), 1.0);
  initializeGenomicElementType("g2", m2, 1.0);

  for (start in seq(from=L0, to=L-(L0+L1), by=(L0+L1)))
      initializeGenomicElement(g2, start, (start+L1)-1);
}
1 {
  sim.addSubpop("p1", N);
  sim.rescheduleScriptBlock(s1, 10*N, 10*N);
}
s1 10 late() {
  sim.treeSeqOutput("./ex2_TS.trees");
}
```

This model sets up a chromosome that consists of genes of length L_1=1 kb, separated by non-coding regions of length L_0=200 kb. The total chromosome length is L=1e8, and 496 genes fit within it. The model uses a mutation rate of r=1e-7 for deleterious mutations that can arise within the genes; no other mutations are modeled. The deleterious mutations are given selection coefficients drawn from a Gamma distribution with mean $-5/N$ and shape parameter α=1, and have a dominance coefficient of h=0.5. A population of size N=10000 is started in generation 1, and the model will run until generation G=10N (the output event, `s1`, is rescheduled dynamically to that generation).

This model took 16818 seconds (4.67 hours) to execute, with peak memory usage of 2532.2 MB. This is not short – it is still a fairly complex model! – but it is far shorter than the equivalent SLiM model including neutral mutations would take.

Having completed that run, we can now execute a Python script that assesses the diversity at each position along the chromosome:

```
python msp-tree-heights.py --tree_file ex2_TS.trees
  --logfile ex2_TS_heights.csv
```

The script (Supplemental S2) does this by walking through the "tree sequence", a key msprime concept (Kelleher et al., 2016), to find the set of trees representing the ancestry of every individual in the final generation at a given position. The average time to the root of the trees for a given position is a metric of the level of diversity at that position; if diversity has been lost due to background selection, the average time to the root will tend to be shorter. The script outputs a CSV file with the mean tree height at each of a set of intervals that span the chromosome (each interval being a region with a uniform pattern of inheritance, unbroken by any recombination event that persisted through to the final generation). This script takes 9.84 seconds.

Finally, a custom R script (Supplemental S3) reads in the CSV file of tree heights and produces a plot (Figure 1). This is the well-known "dip in diversity" plot, realized here through simulation. The R script produces this plot by measuring the distance from each base position to the nearest gene, and tabulating this data into tree height as a function of distance from a gene. As the distance to the nearest gene decreases, diversity dips due to the background selection exerted by the selection against deleterious mutations within the gene.

Example III: True local ancestry mapping

Our third example is of mapping the true local ancestry at every position along the chromosome in a model of admixture between two subpopulations. The SLiM model looks like this:

```
initialize() {
  defineConstant("L", 1e8);   // chromosome length
  initializeTreeSeq();
  initializeMutationRate(0);
  initializeMutationType("m1", 0.5, "f", 0.1);
  initializeGenomicElementType("g1", m1, 1.0);
  initializeGenomicElement(g1, 0, L-1);
  initializeRecombinationRate(1e-8);
}
1 late() {
  sim.addSubpop("p1", 500);
  sim.addSubpop("p2", 500);
  sim.treeSeqRememberIndividuals(sim.subpopulations.individuals);

  p1.genomes.addNewDrawnMutation(m1, asInteger(L * 0.2));
  p2.genomes.addNewDrawnMutation(m1, asInteger(L * 0.8));

  sim.addSubpop("p3", 1000);
  p3.setMigrationRates(c(p1, p2), c(0.5, 0.5));
}
2 late() {
```

```
  p3.setMigrationRates(c(p1, p2), c(0.0, 0.0));
  p1.setSubpopulationSize(0);
  p2.setSubpopulationSize(0);
}
2: late() {
  if (sim.mutationsOfType(m1).size() == 0)
  {
      sim.treeSeqOutput("./ex3_TS.trees");
      sim.simulationFinished();
  }
}
10000 late() {
  stop("Did not reach fixation of beneficial alleles.");
}
```

The `initialize()` callback (the first block of code) sets up tree-sequence recording with a mutation rate of μ=0 and a recombination rate of r=1e-8 along a chromosome of length L=1e8. Although the mutation rate is zero, a mutation type `m1` is defined representing beneficial mutations with a selection coefficient of s=0.1; mutations of this type will be added in script later.

In generation 1 we create two subpopulations, `p1` and `p2`, of 500 individuals each; these are the original subpopulations that will admix. We tell SLiM to remember these individuals forever as ancestors in the tree sequence, because we want them to act as the roots of all trees so that we can establish local ancestry using them. Then we add a beneficial mutation at 0.2L in `p1`, and another at 0.8L in `p2`; the expectation is that by the end of the run all individuals will be recombinants that carry both of these mutations. Finally, we create subpopulation `p3` and tell SLiM that it will be composed entirely of migrants from `p1` and `p2` in equal measure.

By the end of generation 2, subpopulation `p3` has received its offspring generation from `p1` and `p2` as intended, so we can now remove `p1` and `p2` from the model and allow `p3` to evolve. At this stage, all individuals in `p3` are still purebred, having been generated from parents in either `p1` or `p2`, but beginning in generation 3 they will start to mix.

Finally, we have some output and termination code. If both `m1` mutations fix, they are converted to `Substitution` objects by SLiM, and when that is detected the model writes out a final `.trees` file and terminates. If we reach generation 10000 without that happening, the admixture failed, and we stop with an error. This model is conceptually similar to recipe 13.9 in the SLiM manual, but has been converted to use tree-sequence recording, so you can refer to the manual's recipe for additional commentary.

This model runs in 0.504 seconds, with peak memory usage of 69.8 MB; since it tracks only two mutations, and typically terminates by generation 150 or so, it is very quick. We then run a Python script to process its output:

```
python msp-tree-ancestry.py --tree_file ex3_TS.trees
  --logfile ex3_TS_ancestry.csv
```

This script takes ~90 seconds to run, since the processing it does is somewhat involved; most of that time is probably spent in Python, and so a C version of this script using the tskit C front end to msprime would likely be much faster, but in any case it is far faster than the equivalent

SLiM model without tree-sequence recording. As in Example 2, this script walks through the tree sequence and performs an analysis at each site. In this case, however, the analysis is of the mean true local ancestry (the fractional ancestry from subpopulation `p1` versus `p2`) at the given site. This is done by finding the root of each tree at a given position, assessing the subpopulation of origin of that root individual, and averaging those together weighted by the number of descendants from each root. A CSV file is output, as before, and read into an R script to produce a plot. (The use of R for plotting is just personal preference.) The final plot is shown in Figure 2.

The mean true local ancestry at the points where the beneficial mutations were introduced into `p1` and `p2` has to be 100% `p1` and 100% `p2`, respectively, since both beneficial mutations fixed by the end of the run. At other points along the genome there is more variation, but with a general pattern of being more completely admixed at the chromosome ends and middle, with gradations toward the absolute `p1` and `p2` points. Since this is a single run of the model, the pattern is quite stochastic; an average across many runs of this model could produce a smooth plot if desired, and since it takes only a couple of minutes to execute the pipeline here, that would be very quick to do. This method of calculating true local ancestry could be used by any SLiM model with tree-sequence recording, so models with more complex demography, under any scenario of selection and mating, with any recombination map, etc., could just as easily be explored.

Example IV: Neutral burn-in for a non-neutral model

Our final example shows that the flow of information need not go from SLiM into msprime; it can also go from msprime to SLiM. Here we will look at a model of a large population that evolves under neutral dynamics for a burn-in period of G=10N generations and then follows some non-neutral dynamics for a relatively brief time. Running a model this large in SLiM for G generations would take quite a long time, especially with neutral mutations included, but there is no need to do that. Instead, we can use msprime to generate the final state of the burn-in period using the coalescent, and output the result as a tree-sequence file. We can then read that file in to SLiM, instantiating msprime's simulation at the moment when it completed with full tree sequence ancestry information, and run in SLiM for the remaining generations of non-neutral dynamics (in this case, executing a two-locus selected sweep from new beneficial mutations). All of this can be done without ever actually simulating the neutral mutations; those can be overlaid at the end, as in Example 1.

We begin with the call to msprime to create the burned-in simulation state (as in Example 1), this time with a population size of N=1e5 individuals (and therefore 2N=2e5 samples), again with L=1e8 and r=1e-8, but this time with μ=0 since we are deferring the addition of mutations:

```
import msprime
ts = msprime.simulate(sample_size=200000, Ne=200000, length=1e8,
  recombination_rate=1e-8, mutation_rate=0)
ts.dump("ex1_msprime.trees")
```

This runs in XXXXX seconds. Once that has completed, we run a SLiM model that reads in the resulting `.trees` file and runs the non-neutral remainder of the model:

model code to be added once we can load .trees files in SLiM…

This model begins where the msprime coalescent simulation left off. It then sequentially introduces two beneficial mutations into the population, one onto the background of the other, both with large positive selection coefficients (s=0.5). It then runs until both mutations are either fixed or lost, and outputs a new `.trees` file containing the final state of the model. Even with these large positive selection coefficients, it is of course possible for the beneficial mutations to be lost due to drift; one could make this simulation conditional on fixation, using the techniques shown in the SLiM manual (Haller and Messer, 2016), but that is beyond the scope of this example. A run of the model has been selected for presentation here that did fix both mutations; it completed in X generations, with a runtime in SLiM of X seconds.

The last step, potentially, could be to overlay neutral mutations upon the final tree sequence, as in Example 1, if that were desirable for whatever analysis might follow. On the other hand, it might not be necessary, since some analyses can be done directly using the patterns of inheritance encoded within the tree sequence, as we have shown for diversity (Example 2) and true local ancestry (Example 3). Since the overlay of neutral mutations is optional, and in any case would follow Example 1 exactly, we will not perform that final step here. Instead, we will use the SVG plotting functionality provided by msprime to show the trees at a few sites along the chromosome (Figure 3). These illustrate how the sequential sweeps affected the diversity at different sites, as well as providing a visual illustration of the concept that underlies tree sequence recording.

Discussion

Forward simulation is increasingly used in population genetics and evolutionary biology, but a desire for increased realism – full genome sequences, large populations, complex demography, explicit space, ecological and spatial interactions, and so forth – means that computational resources are often inadequate for the simulations researchers want to run. Although increases in processor speed and improvements in software mitigate this to some extent, the inefficiency of forward simulation is hindering progress in many research areas.

We believe that the tree-sequence recording method developed by Kelleher et al. (2018) provides a path forward, by allowing both the execution of forward simulations and the analysis of their results to be sped up by well over an order of magnitude in many common scenarios. We have therefore integrated support for tree-sequence recording in the popular SLiM forward simulation software package; tree-sequence recording can now be enabled in any simulation, and the results output to a `.trees` file for the overlaying of neutral mutations after the fact, or for downstream analysis of the simulation that relies upon the tree-sequence information for more efficient calculation of statistics. These output files can also be loaded into Python with the msprime package for any custom analysis or processing desired.

We have also extended the tree-sequence recording method of Kelleher et al. (2018) to allow the recording and output of mutations (usually non-neutral mutations) that arise during forward simulation. This allows the `.trees` file to provide a more complete snapshot of the simulation state, and allows Python-based post-simulation analysis to include information about the particular mutations that occurred. The `.trees` file can now be used as an input format, as well

as an output format, in SLiM, providing a way to load in simulation state that includes information about ancestry and history.

We showed four examples in this paper. The first example, of a simple neutral model, showed that tree-sequence recording requires just a single line of code in SLiM to enable it, plus a few modifications to turn off neutral mutations, and results in more than a 40-fold speedup compared to the equivalent model without tree-sequence recording, as well as a two-thirds decrease in memory usage. The second example illustrated the use of recorded tree sequences to estimate the "dip in diversity" seen at neutral loci near genes due to background selection, showing how to conduct such post-simulation analysis in Python with a simple custom script. The third example was of mapping the mean true local ancestry along the chromosome in a model of the admixture of two subpopulations; this is possible in SLiM without tree sequences, but would be prohibitively slow for a chromosome of the length that was quickly and easily modeled here. Finally, the fourth example illustrated the use of msprime's ability to simulate tree sequences with the coalescent, leveraging that to accelerate the neutral burn-in period of a SLiM model that then enters into non-neutral dynamics from that initial neutral equilibrium.

All of the examples after the first were illustrated only with tree-sequence recording; no comparison to the equivalent models without tree-sequence recording was shown. This is because running those equivalent models would have taken an extremely long time, if it turned out to be possible to run them at all on the hardware we used. This is, then, a vivid illustration of the core point: tree-sequence recording makes simulations possible that would previously have been, if not literally impossible, then at least completely impractical. This should open up new horizons for exploration, for many researchers. The software used here – SLiM, msprime, Python, R – is all free and open-source. We hope that the practical examples we have provided here will raise the level of awareness in evolutionary biology regarding this exciting new method.

Acknowledgements

This work was supported by funds from the College of Agriculture and Life Sciences at Cornell University to PWM; by funding from the Sloan Foundation and the NSF (under DBI-1262645) to PLR; and by the Wellcome Trust (grant 100956/Z/13/Z) to Gil McVean for JK. ***[Modify these as needed, I just took them from recent prior work…]***

References

Haller, B.C., and Messer, P. W. (2016). SLiM: An Evolutionary Simulation Framework. URL: http://benhaller.com/slim/SLiM_Manual.pdf

Haller, B.C., and Messer, P.W. (2017). SLiM 2: Flexible, interactive forward genetic simulations. *Molecular Biology and Evolution 34*(1), 230–240. DOI: http://dx.doi.org/10.1093/molbev/msw211

Kelleher, J, Etheridge, A.M., and McVean, G. (2016). Efficient coalescent simulation and genealogical analysis for large sample sizes. *PLoS Computational Biology 12*(5): e1004842. DOI: https://doi.org/10.1371/journal.pcbi.1004842

Kelleher, J., Thornton, K.R., Ashander, J., and Ralph, P.L. (2018). Efficient pedigree recording for fast population genetics simulation. bioRxiv, DOI: http://dx.doi.org/10.1101/248500

[lots more refs needed; I am not using a reference manager, just add cites here or point me at papers and I'll add them…]

Figures

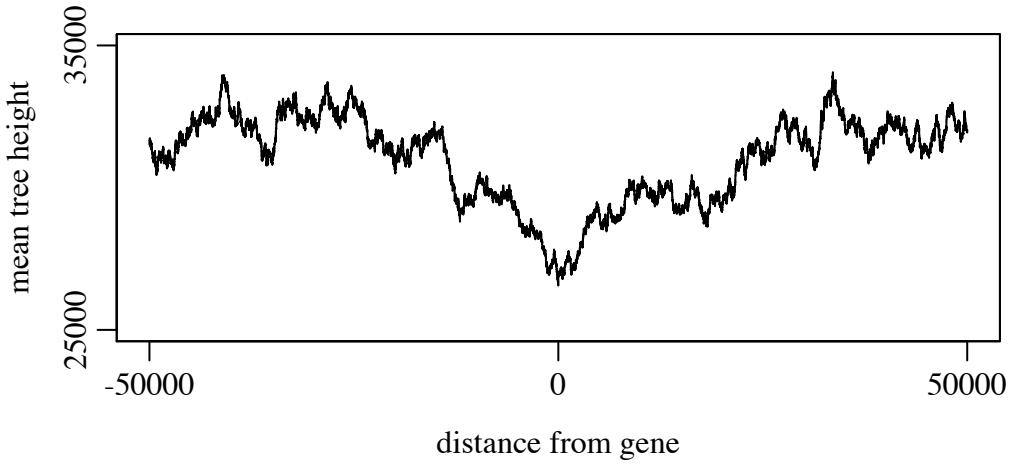

Figure 1. Mean diversity (as measured by mean tree height) as a function of distance from the nearest gene (Example 2). The center of the x-axis represents a distance of zero, immediately adjacent to a gene; moving away from the x-axis center to the left/right represents moving away from the nearest gene to the left/right respectively. The pattern of decreased diversity near a gene is the "dip in diversity" due to background selection.

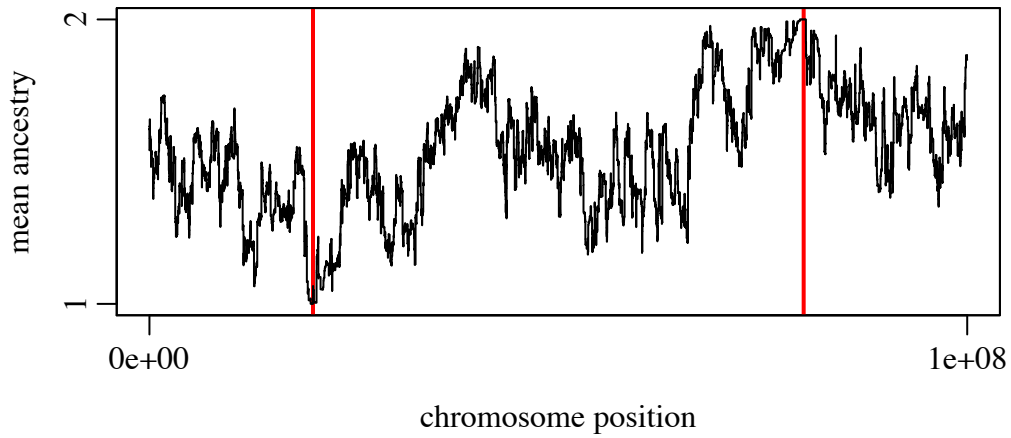

Figure 2. Mean true local ancestry at each position along the chromosome (Example 3). The red lines indicate the positions at which beneficial mutations were originally introduced into `p1` and `p2`. The beneficial mutations, which both fixed, are points where the true

local ancestry is 100% `p1` or `p2`. True local ancestry regresses toward equal admixture with increasing distance from those fixed points.

fill this in once the example is done…

Figure 3. Trees at a few selected sites after neutral burn-in and then two overlapping beneficial sweeps (Example 4). (A) The tree at position 2e7, the site of the first beneficial mutation introduced. (B) The tree at position 3e7, the site of the second beneficial mutation introduced. (C) The tree at position 9e7, relatively far from the introduced beneficial mutations on the chromosome of length 1e8. ***[Could have another three panels, I suppose, showing the same positions after neutral burn-in, for comparison. Probably overkill.]***

Supplemental

S1: Python code for the `msp-add-mutation.py` script.

S2: Python code for the `msp-tree-heights.py` script.

S3: R code to produce the "dip in diversity" plot, Figure 1, for Example 2.

S4: R code to produce the "true local ancestry plot", Figure 2, for Example 3.

All examples are shared on GitHub at XXXXX ***[should this perhaps go under SLiM-Extras?]*** with the full SLiM, Python, and R code.