[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
Big Data Institute, Li Ka Shing Centre for Health Information and Discovery
University of Oxford
Oxford, OX3 7FZ, UK

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

(word count: 238)

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 recently developed tree-sequence recording method (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 genetic 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 tree-sequence recording method 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 this method. We here present several examples of the practical application of this method, which we believe opens up new horizons of possibility in the modeling of evolutionary processes.

Keywords

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

Introduction

(word count: 1210)

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 (Fournier-Level et al., 2016; Cotto et al., 2017; Matz et al., 2018; Ryan et al., 2018), for discovering the ecological and evolutionary mechanisms that led to present-day genomic patterns in a species (Enard et al., 2014; Nowak et al., 2014; Arunkumar et al., 2015; Patel et al., 2018), for testing or validating empirical and statistical methods (Haller and Hendry, 2013; Caballero et al, 2015; Ewing et al., 2016; Haller and Messer, 2017), and for exploring theoretical ideas about evolution (Haller et al., 2013; Assaf et al., 2015; Mafessoni and Lachmann, 2015; Champer et. al, 2018), among other purposes. 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 including 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 those 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, as have performance improvements due to improved forward simulation software (Messer, 2013; Thornton 2014; Haller and Messer, 2017), 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 recently developed "pedigree recording" or "tree-sequence recording" method (Kelleher et al., 2018) is potentially transformative. Kelleher et al. (2018) show, counter-intuitively, that by recording all ancestry information for the entire population, simulations can run orders of magnitude more quickly. These gains in efficiency are made possible by the succinct tree sequence data structure (or "tree sequence", for brevity) that lies at the heart of the msprime coalescent simulator (Kelleher et al., 2016). The tree sequence data structure is a concise encoding of the correlated genealogies resulting from evolution in sexually reproducing populations. The key insight of the pedigree-recording method for forward simulations is that precisely the same tree sequence data structure can be used to model the direct parent–child relationships that occur from generation to generation. This data structure will then record who each individual inherited each section of chromosome from, for every individual that ever lived. There is a massive amount of redundancy in this information, since many of the individuals simulated in the past will leave no descendants in the extant population; such redundancy is removed by periodically "simplifying" the tree sequence. This combination – the tree sequence data structure and an efficient algorithm for simplifying it – allows complete genealogies to be recorded efficiently in forward simulations for the first time.

The record of ancestry information produced by tree-sequence recording is important 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 can be accurately overlaid after forward simulation has completed. This provides an immense efficiency benefit, since neutral mutations only need to be added along those branches of the ancestry tree that are represented in the final simulation generation; all other ancestral branches, which typically comprise the vast majority of the full evolutionary tree, can be ignored since they do not contribute to the final 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 sequence 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 sequence 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 quite important.

A third reason is that a record of the pattern of ancestry and inheritance in a population can itself be useful. The tree sequence from a simulation can be loaded into Python, and can then be analyzed and even manipulated for 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-packaged ways of using tree sequences. Instead, new ways of using tree sequences 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 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 enable tree-sequence recording in any SLiM model, and then to output the recorded tree sequence 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 while preserving information about ancestry. 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 method. 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 mutations. Our third example will be a model of admixture between two subpopulations, showing how to use the recorded tree sequence to calculate the mean true local ancestry at every position along the chromosome. Finally, the fourth example illustrates the use of msprime to very efficiently add a "neutral burn-in" history to a completed SLiM simulation by coalescing the simulation's initial population back in time, a process we call "recapitation". These 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

(word count: 91 + 66 + 97 + 139 + 1267 + 91 + 135 + 713 + 30 + 354 + 504 + 946 = 4433)

All examples were executed on a MacBook Pro (2.9 GHz Intel Core i7, 16 GB RAM) running macOS 10.13.4, using Python 3.4.8, R 3.4.3, and prerelease versions of SLiM 3.0 and msprime 0.6.0. All reported times were measured with the Un*x tool `/usr/bin/time` (summing the reported user time and system time). Peak memory usage for SLiM runs was assessed with SLiM's `-m` command-line option. The full source code for all examples, including timing and plotting code that is omitted here, may be found at https://github.com/bhaller/SLiMTreeSeqPub.

Example I: A simple neutral model

Our first example is a model of neutral evolution on a chromosome of length $L = 10^8$ base positions, with mutation rate $\mu = 10^{-7}$ and recombination rate $r = 10^{-8}$ (both expressed as the event probability per base per generation), in a panmictic population of size $N = 500$, running for a duration of $10N = 5000$ non-overlapping generations. The SLiM model for this is very simple:

```
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.slimbinary", 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 220.7 seconds to run, and reached a peak memory usage of 385.8 MB.

This model is easily recast to use tree-sequence recording:

```
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. 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 simulation has completed, neutral mutations are overlaid upon the saved tree sequence. The full model – running the SLiM model and then doing the final mutation overlay step – can be executed with a simple Python script:

```
import subprocess, msprime

# Run the SLiM model
subprocess.check_output(["slim", "-m", "-s", "0", "./ex1_TS.slim"])
```

```
# Overlay neutral mutations
ts = msprime.load("./ex1_TS.trees")
mutated = msprime.mutate(ts, rate=1e-7, random_seed=1, keep=True)
mutated.dump("./ex1_TS_overlaid.trees")
```

This script 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 the model without tree-sequence recording; 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. The total time to execute this Python code is 4.29 seconds, more than 50 times faster than the model without tree-sequence recording. Most of the runtime (4.04 seconds) is spent running the SLiM model; the final mutation overlay by msprime is extremely fast. The peak memory usage during the SLiM run is 136.3 MB, just over one-third of the memory usage of the model without tree-sequence recording. Tree-sequence recording can often reduce memory usage, since the tree sequence data structure is quite compact compared to SLiM's in-memory representation of the neutral mutations that would be segregating in such a model.

The speedup produced by this 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 very short chromosome the overhead of tree-sequence recording can outweigh the savings from omitting neutral mutations. It may thus be worthwhile to check the runtime of both methods to obtain the best performance for a particular model; however, the time required to process the results of a simulation should also be taken into account, as the tree-sequence format can provide significant advantages during post-simulation analysis.

To further illustrate the performance benefits of tree-sequence recording, we conducted a set of timing comparisons between SLiM without tree-sequence recording, SLiM with tree-sequence recording, and msprime's coalescent simulation method. These comparisons involved essentially the same model as shown above: a neutral panmictic model of diploids with non-overlapping generations, with a population size $N = 500$, recombination rate $r = 10^{-8}$ per base position per generation, and mutation rate $\mu = 10^{-7}$ per base position per generation. The chromosome length L was varied over $\{10^5, 10^6, 10^7, 10^8, 10^9, 10^{10}\}$, with ten runs of each model at each value of L using different random number seeds. To provide a fair comparison to the msprime coalescent, instead of running for $10N$ generations as above (which is just a rough heuristic), the forward simulations were run to the mean number of generations to coalescence for the model with the given value of L (see below for details). The msprime coalescent was run with a final sample size equal to the full population size ($n = N$), and with a much smaller sample size ($n = N/100$); in both cases, $N_e = N$ was used.

The average times obtained are shown in Figure 1. As L increased, the benefit of tree-sequence recording compared to SLiM without tree-sequence recording became increasingly large, topping out at a performance improvement of more than two orders of magnitude for

$L = 10^9$ and $L = 10^{10}$. The msprime coalescent runs were generally even faster than the tree-sequence recording method; however, at $L = 10^{10}$ the time for SLiM with tree-sequence recording was very close to msprime's time, and it appears likely that for even larger values of L, SLiM with tree-sequence recording would be the faster method. Since one of us (Kelleher) is an author on the msprime project, we can explain this: msprime's coalescent simulation algorithm contains some elements that are $O(L^2)$ in complexity, but with a very small constant multiplier such that at typical values of L this term contributes little to the overall runtime. With very large values of L, however, this $O(L^2)$ term begins to dominate, allowing SLiM with tree-sequence recording, which apparently has a lower order of algorithmic complexity in L, to become faster. This may be chiefly of theoretical interest, however, since $L = 10^{10}$ is already a very long chromosome (approximately three times the length of the full human genome). It is also noteworthy that the msprime coalescent is only marginally faster for a sample of $n = N/100$ than for a full population sample of $n = N$; as more samples are added to a gene tree, the new samples tend to attach to already existing branches quite quickly (Kingman, 1982), explaining this result.

Although the coalescent remains faster for most practical purposes, it can only be used in a few simple scenarios such as this; for models that require forward simulation, tree-sequence recording offers large performance benefits over more traditional forward simulation techniques. 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 (Wakeley et al., 2012; Bhaskar et al., 2014) – one such condition being a sample size that is large relative to the population size, as is the case for our $n = N$ msprime runs here. Forward simulation may therefore be preferable to obtain exact results under such conditions.

To validate the procedure used for these timing comparisons, we compared the mean time to the most recent common ancestor (TMRCA) between the $L = 10^{10}$ runs for SLiM with tree-sequence recording and the msprime coalescent (for the $n = N$ msprime case). A Welch's unequal-variance t-test showed that the mean TMRCAs between the two simulation methods were not significantly different ($p = 0.7791$), indicating that the simulations were indeed comparable.

Explanation of one issue was put off above: the run length of the forward simulations. To provide a fair comparison between SLiM and msprime, we ran each SLiM simulation out to the mean number of generations to coalescence for that simulation's value of L. Averaged across multiple runs, SLiM's runtime to this point ought to be essentially the same as its runtime to the point of actual coalescence. The mean generations to coalescence for each value of L was determined by running the same model as above, with an additional "coalescence detection" feature enabled by passing `checkCoalescence=T` to `initializeTreeSeq()`; calls to `sim.treeSeqSimplify()` and then `sim.treeSeqCoalesced()` were then made every 100th generation to detect coalescence. Since checking every 100th generation overestimates the coalescence time by an average of 50 generations, 50 was subtracted from each measured coalescence time. For each value of L, 500 replicates with different random number seeds were run. The mean and other summary statistics for each value of L are shown in Table 1. Note that the relative standard error of the mean is below 2.5% in all cases, and below 1% for the largest values of L, so any error in the estimated mean coalescence times should not substantially affect

the results shown in Figure 1. [OMIT THIS NEXT BIT?] (It might seem simpler to have used SLiM's coalescence-detection feature to terminate the timing comparison runs directly; however, coalescence detection requires tree-sequence recording to be enabled, making it unusable in the SLiM runs that did not use tree-sequence recording. Also, coalescence detection entails additional performance costs that would have skewed the comparison; we expect that most models will not use coalescence detection, so its overhead should not be included in the timing comparison.)

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 more often have functional effects, linked selection (background selection and hitchhiking) 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 (Charlesworth et al. 1993; Hudson 1994; Sattath et al., 2011; Elyashiv et al., 2016). 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 $L = 10^8$ bases, and 496 genes fit within it. The model uses a mutation rate of $r = 10^{-7}$ 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 $\alpha = 1$, and have a dominance coefficient of $h = 0.5$. A population of size $N = 10000$ is started in generation 1, and the model runs until generation $G = 10N$ (the output event, `s1`, is rescheduled dynamically to that generation).

We can run this model and then conduct post-run analysis with a Python script [MAYBE SOMEBODY GOOD IN PYTHON CAN MAKE THIS SCRIPT SHORTER?]:

```
import subprocess, msprime, statistics, matplotlib.pyplot

# Run the SLiM model
subprocess.check_output(["slim", "-m", "-s", "0", "./ex2_TS.slim"])

# Load the .trees file and measure the tree height at each base position
height_for_pos = []
ts = msprime.load("./ex2_TS.trees")
for tree in ts.trees():
    mean_height = statistics.mean([tree.time(root) for root in tree.roots])
    height_for_pos += [mean_height] * int(tree.interval[1] -
tree.interval[0])

# Convert heights along the chromosome into heights at distances from a gene
min_height = min(height_for_pos)
height_for_pos = [x - min_height for x in height_for_pos]
L, L0, L1 = int(1e8), int(200e3), int(1e3)
height_left, height_right = [], []
gene_starts = list(range(L0, L - (L0 + L1) + 1, L0 + L1))
gene_ends = [x + L1 - 1 for x in gene_starts]

for distance in range(1, L0 // 4 + 1):
    left_heights = [height_for_pos[x - distance] for x in gene_starts]
    height_left.append(statistics.mean(left_heights))
    right_heights = [height_for_pos[x + distance] for x in gene_ends]
    height_right.append(statistics.mean(right_heights))

height_for_distance = height_left[::-1] + height_right
distances = list(range(-L0//4, 0)) + list(range(1, L0//4 + 1))

# Make a simple plot
matplotlib.pyplot.plot(distances, height_for_distance)
matplotlib.pyplot.show()
```

The first line after the import statement runs the SLiM model; this took 15501 seconds (4.30 hours) to execute. This is not short – it is still a fairly complex model! – but it is far shorter than the alternative. That alternative, a SLiM model without tree-sequence recording and including neutral mutations in the non-coding regions, is roughly estimated to take 298072 seconds, or 82.8 hours, by extrapolation from the time to run 100 generations after the model had run for an hour. This appears to be a conservative estimate, since the model was still slowing down (presumably because it had not yet reached mutation-selection balance); but ignoring that, the use of tree-sequence recording here results in a relatively modest speedup of 19.2 times. This makes sense, since the model with tree-sequence recording is still simulating a very large number of segregating deleterious mutations; the additional bookkeeping involved in also tracking all of the neutral mutations is relatively modest. However, it's worth noting that the final result from this alternative model would provide far less statistical power, since inference from it would be based only upon the observed pattern of neutral mutations in one run, rather than the actual pattern of ancestry at each chromosome position; to provide the same power, this alternative model would likely have to be run many times.

The rest of the code conducts post-run analyses. First, the `.trees` file from the SLiM run is read in, and a vector containing the mean tree height at each base position is constructed (`height_for_pos`). The mean tree height is a metric of the time to the most recent common

ancestor at a given base position, and thus of diversity at that base position; if diversity has been lost due to background selection, the mean tree height will tend to be shorter. This vector is constructed by walking through the tree sequence to find the set of trees representing the ancestry of every individual in the final generation at a given position. An aside: there can be a set of trees for a given position, rather than just a single tree, because an important property of trees generated by forward-time simulations is that there will often not be a single common ancestor of all the leaves. Depending on the model and how many generations it is run for, there may not be sufficient time for a single common ancestor of the entire population to exist at a given position on the genome. In msprime this is modelled by allowing trees to have multiple roots; a tree with n leaves may have 1 to n roots. Each root represents the most recent common ancestor of some subset of the extant population at that location on the genome. Since the model here ran for $10N$ generations, we can hope that it has coalesced at most or all positions; but unless a model is explicitly run out to coalescence, it is always possible that multiple roots will exist, and so robust code ought to handle that case by looping over the roots for each tree as we do here.

These mean tree heights along the chromosome are then converted to tree heights at distances from the nearest gene (`height_for_distance`), taking into account the somewhat complex genetic structure of the model. Finally, the relationship between distance to the nearest gene and tree height is plotted. These analyses took 87.5 seconds to complete. Note that neutral mutations were never simulated at all; the analysis is based upon the tree sequence itself, not upon the distribution of neutral mutations.

A plot of the results can be seen in Figure 2 (this is a production-quality plot from R, not the simple plot produced above). This is the well-known "dip in diversity" plot, realized here through simulation. As the distance to the nearest gene decreases, diversity dips due to the background selection exerted by 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);
  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 sets up tree-sequence recording with a mutation rate of $\mu = 0$ and a recombination rate of $r = 10^{-8}$ along a chromosome of length $L = 10^8$. 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 generation 1.

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, with `treeSeqRememberIndividuals()`, because we want them to act as the roots of all recorded 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 (Haller and Messer, 2016), but has been converted to use tree-sequence recording, so you can refer to the manual's recipe for additional commentary.

We can run this model from a Python script and do post-run analysis, as we did in Example 2:

```
import subprocess, msprime, matplotlib.pyplot

# Run the SLiM model
subprocess.check_output(["slim", "-m", "-s", "0", "./ex3_TS.slim"])
```

```
# Load the .trees file and assess true local ancestry
starts, ends, subpops = [], [], []
ts = msprime.load("./ex3_TS.trees")
for tree in ts.trees(sample_counts=True):
    subpop_sum, subpop_weights = 0, 0
    for root in tree.roots:
        leaves_count = tree.num_samples(root) - 1
        subpop_sum += tree.population(root) * leaves_count
        subpop_weights += leaves_count
    starts.append(tree.interval[0])
    ends.append(tree.interval[1])
    subpops.append(subpop_sum / float(subpop_weights))

# Make a simple plot
x = [x for pair in zip(starts, ends) for x in pair]
y = [x for x in subpops for _ in (0, 1)]
matplotlib.pyplot.plot(x, y)
matplotlib.pyplot.show()
```

The first line after the import statement runs the SLiM model, which completes in just 0.389 seconds, with peak memory usage of 53.1 MB; since it tracks only two mutations, and typically terminates by generation 150 or so, it is very quick.

The equivalent SLiM model to achieve true local ancestry mapping without tree-sequence recording has to model a mutation at each base position, as can be seen in recipe 13.9 in the SLiM manual (Haller and Messer, 2016). A direct comparison is not possible, because recipe 13.9 scaled up to a chromosome length of $L = 10^8$ would take an estimated 7.2 days to run, and worse, would require 8.1 TB of memory. Those estimates are derived from the pattern of performance observed for recipe 13.9 with $L = 5\times10^5$, $L = 10^6$, and $L = 2\times10^6$ (the upper limit on our test machine due to memory usage), extrapolated out to $L = 10^8$. Implementing this model with tree-sequence recording therefore reduces the runtime by a factor of more than 1.35 million, and reduces the memory usage by a factor of more than 160,000.

The post-run analysis in this script takes 92.3 seconds to run, since the processing it does is somewhat involved. 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 roots for the tree at a given position, assessing the subpopulations of origin of those root individuals, and averaging those together weighted by the number of descendants from each root. A simple plot is then produced (the first plotting line interleaves `starts` and `ends` to produce an `x` vector, and the second plotting line duplicates the values in `subpops` to produce a `y` vector).

The final plot of true local ancestry by chromosome position is shown in Figure 3 (again, this is a production plot made in R). 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 illustrates a solution to the problem of neutral burn-in. Very often, one wishes to execute a non-neutral forward simulation from an equilibrium neutral state, and the simulation needed to generate that equilibrium neutral state, often called the model "burn-in", can take quite a long time – often much longer than it takes to execute the non-neutral portion of the simulation, in fact. For a model with a long chromosome or large population size, this burn-in can be so long as to limit the practical scale of the simulations that can be conducted. The key realization here is that the vast majority of work done in simulating this burn-in period is unnecessary. All of the evolutionary branches that go extinct are irrelevant; all that matters is the pattern of neutral mutations present in the genomes of the individuals that exist at the end of the simulation. With tree-sequence recording, one can model just that, saving a tremendous amount of computation.

Here we will look at a model of a large population that evolves under neutral dynamics until coalescence (the neutral burn-in), and then follows some non-neutral dynamics for a relatively brief time. Running the burn-in period for a model this large in SLiM would take quite a long time, but instead, we can run the model forward from an initial state that is conceptually after burn-in, and then use msprime to generate the coalescent history for the initial individuals of the forward simulation, a method we call "recapitation". This can be done without ever actually simulating the neutral mutations; but if neutral mutations are desired as an end product of the simulation, they can be overlaid at the end as in Example 1.

We begin with the SLiM model for the non-neutral portion of the simulation. This model entails the simultaneous introduction and sweep to fixation of two beneficial mutations. For simplicity, we will select a run of the model that happens to result in fixation, rather than using a recipe that is conditional upon fixation; the random number seed set here should produce that outcome, for SLiM 3.0. The SLiM model:

As in previous examples, we will run this model from a Python script that also does post-run analysis:

The first line after the import is, again, a run of the SLiM model, which completes in XXXXX seconds. After the SLiM model completes, the next lines perform the recapitation. This process works backwards from the tree sequence information recorded by SLiM, constructing a full coalescent history for all of the individuals alive at the end of the run. Since the two selective sweeps eliminated most of the genetic diversity from the population as it existed at the beginning of forward simulation, this coalescence requires very little work – much less than even a normal

coalescent simulation for this population size would require. In this example run, the process took XXXXX seconds.

The best alternative to recapitation would probably be to run the neutral burn-in using msprime's coalescent simulation, and then load the result into SLiM as the starting point for non-neutral dynamics. As a comparison, then, an equivalent coalescent simulation with msprime ($n = N = N_e = 100000$, $L = 10^8$, $r = 10^{-8}$, $\mu = 10^{-7}$) took 9299.9 seconds, which is 2.58 hours. Recapitation therefore sped up the burn-in process in this example by a factor of roughly XXXXX.

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, this 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 have not performed that final step here. [DO WE WANT TO DO THIS NEXT THING, OR IS IT OVERKILL?] Instead, the script above uses the SVG plotting functionality provided by msprime to show the trees at a few sites along the chromosome (Figure 4). These illustrate how the two selective sweeps affected the diversity at different sites, as well as providing a visual illustration of the concept that underlies tree sequence recording.

Constructing a burn-in history with recapitation only works if the burn-in period is completely neutral. If a non-neutral burn-in to equilibrium is needed, the best approach is probably to run the burn-in period in SLiM with tree-sequence recording turned on and neutral mutations turned off, so that the cost of simulating the neutral mutations during burn-in is avoided. If a neutral burn-in is desired, but the neutral mutations are then needed by the non-neutral portion of the simulation (perhaps because some of the neutral mutations become non-neutral due to an environmental change), one might simulate the burn-in period with the coalescent in msprime, and then save the result as a `.trees` file; one could then read that `.trees` file into SLiM to provide the initial state for further simulation. These techniques go beyond what we have space to illustrate here, but the manual for SLiM 3.0 provides further recipes illustrating the use of tree-sequence recording. Since it is possible to move simulation data, with full ancestry records, back and forth between msprime and SLiM, one can imagine many ways to leverage the strengths of the coalescent and forward simulation, while avoiding their respective weaknesses.

Discussion

(word count: 686)

Forward simulation is increasingly used in population genetics and evolutionary biology, but a desire for increased scale and realism 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 hinders progress in many research areas.

We believe that the tree-sequence recording method (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 tremendously in many common scenarios. We have therefore integrated support for tree-sequence recording into 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 that can be loaded into Python with the msprime package for further simulation or analysis.

We have also extended the tree-sequence recording method to allow the recording and output of 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 ancestry.

We showed four examples of the power of tree-sequence recording in this paper. The first example, of a simple neutral model, showed how to enable tree-sequence recording with a few trivial modifications to a SLiM model's script. The second example illustrated the use of recorded tree sequences in post-simulation analysis in Python to estimate the "dip in diversity" seen at neutral loci near genes due to background selection. The third example mapped the mean true local ancestry along the chromosome in a model of the admixture of two subpopulations, again using post-simulation Python analysis. Finally, our fourth example illustrated the use of msprime to "recapitate" a SLiM run, using the coalescent to construct a neutral burn-in period after the completion of forward simulation. All of these examples illustrated the large performance benefits that can come from tree-sequence recording. Indeed, for very large simulations our timing comparison indicated that the speedup due to tree-sequence recording can exceed two orders of magnitude, and can put the performance of forward simulation on par with msprime's coalescent simulation.

Tree-sequence recording is not a panacea. Models that do not involve neutral mutations will realize no speed benefit from tree-sequence recording; in fact, they will run more slowly, since the overhead of recording will not be compensated by eliminating neutral mutation simulation. Models that involve a very high recombination rate relative to the mutation rate may also not see a speed benefit from tree-sequence recording, since tracking the recombination edges will be so time-consuming; informal tests indicate that this occurs when the recombination rate is more than three orders of magnitude larger than the mutation rate, however, so it may not be a practical concern for most models. Even if simulation performance is not improved, the ancestry information provided by the tree sequence may speed up analysis or provide additional statistical power. In short, whether and how to use tree-sequence recording will depend closely upon the details of both the model and the planned post-run analysis.

Although tree-sequence recording is not appropriate in every model, the examples we have presented showed that the performance gains it provides can make simulations possible that would previously have been beyond reach, opening up new horizons for exploration. The software packages used here – SLiM, msprime, Python, R – are all free and open-source, and the examples and analyses shown here are all available on GitHub at the link given above. We hope that the practical examples we have provided will raise the level of awareness among evolutionary biologists 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.

References

Arunkumar, R., Ness, R.W., Wright, S.I., and Barrett, S.C. (2015). The evolution of selfing is accompanied by reduced efficacy of selection and purging of deleterious mutations. *Genetics 199*(3), 817-829.

Assaf, Z.J., Petrov, D.A., and Blundell, J.R. (2015). Obstruction of adaptation in diploids by recessive, strongly deleterious alleles. *PNAS 112*(20), E2658-E2666.

Bhaskar, A., Clark, A.G., and Song, Y.S. (2014). Distortion of genealogical properties when the sample is very large. *PNAS 111*(6), 2385–2390.

Caballero, A., Tenesa, A., & Keightley, P.D. (2015). The nature of genetic variation for complex traits revealed by GWAS and regional heritability mapping analyses. *Genetics 201*(4), 1601–1613.

Charlesworth, B., Morgan, M.T., and Charlesworth, D. (1993). The effect of deleterious mutations on neutral molecular variation. *Genetics 134*(4), 1289–1303.

Champer, J., Liu, J., Oh, S.Y., Reeves, R., Luthra, A., Oakes, N., Clark, A.G., and Philipp W. Messer, P.W. (2018). Reducing resistance allele formation in CRISPR gene drive. PNAS (early access), 1–6. DOI: https://doi.org/10.1073/pnas.1720354115

Cotto, O., Wessely, J., Georges, D., Klonner, G., Schmid, M., Dullinger, S., Thuiller, W., and Guillaume, F. (2017). A dynamic eco-evolutionary model predicts slow response of alpine plants to climate warming. *Nature Communications 8*, 15399.

Elyashiv, E., Sattath, S., Hu, T. T., Strutsovsky, A., McVicker, G., Andolfatto, P., Coop, G. & Sella, G. (2016). A genomic map of the effects of linked selection in *Drosophila*. *PLoS Genetics 12*(8), e1006130.

Ewing, G.B., and Jensen, J.D. (2016). The consequences of not accounting for background selection in demographic inference. *Molecular Ecology 25*(1), 135–141.

Enard, D., Messer, P.W., and Petrov, D.A. (2014). Genome-wide signals of positive selection in human evolution. *Genome Research 24*(6), 885–895.

Fournier-Level, A., Perry, E.O., Wang, J.A., Braun, P.T., Migneault, A., Cooper, M.D., Metcalf, C.J.E., and Schmitt, J. (2016). Predicting the evolutionary dynamics of seasonal adaptation to novel climates in *Arabidopsis thaliana*. *PNAS 113*(20), E2812–E2821.

Haller, B.C., and Hendry, A.P. (2013). Solving the paradox of stasis: Squashed stabilizing selection and the limits of detection. *Evolution 68*(2), 483–500.

Haller, B.C., R Mazzucco, R., and Dieckmann, U. (2013). Evolutionary branching in complex landscapes. *American Naturalist 182*(4), E127-E141.

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). asymptoticMK: A web-based tool for the asymptotic McDonald–Kreitman test. *G3: Genes, Genomes, Genetics 7*(5), 1569–1575.

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

Hudson, R.R. (1994). How can the low levels of DNA sequence variation in regions of the *Drosophila* genome with low recombination rates be explained? *PNAS 91*(15), 6815–6818.

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

Kingman, J.F.C. (1982). On the genealogy of large populations. *Journal of Applied Probability 19*, 27–43.

Mafessoni, F., and Lachmann, M. (2015). Selective strolls: fixation and extinction in diploids are slower for weakly selected mutations than for neutral ones. *Genetics 201*(4), 1581–1589.

Matz, M.V., Treml, E.A., Aglyamova, G.V., and Bay, L.K. (2018). Potential and limits for rapid genetic adaptation to warming in a Great Barrier Reef coral. *PLoS Genetics 14*(4), e1007220.

Messer, P.W. (2013). SLiM: Simulating evolution with selection and linkage. *Genetics 194*(4), 1037–1039.

Nowak, M.D., Haller, B.C., and Yoder, A.D. (2014). The founding of Mauritian endemic coffee trees by a synchronous long-distance dispersal event. *Journal of Evolutionary Biology 27*(6), 1229–1239.

Patel, R., Scheinfeldt, L.B., Sanderford, M.D., Lanham, T.R., Tamura, K., Platt, A., Glicksberg, B.S., Xu, K., Dudley, J.T., and Kumar, S. (2018). Adaptive landscape of protein variation in human exomes. Molecular Biology and Evolution (early access), msy107. DOI: https://doi.org/10.1093/molbev/msy107

Ryan, S.F., Deines, J.M., Scriber, J.M., Pfrender, M.E., Jones, S.E., Emrich, S.J., and Hellmann, J.J. (2018). Climate-mediated hybrid zone movement revealed with genomics, museum collection, and simulation modeling. *PNAS 115*(10) E2284-E2291.

Sattath, S., Elyashiv, E., Kolodny, O., Rinott, Y., and Sella, G. (2011). Pervasive adaptive protein evolution apparent in diversity patterns around amino acid substitutions in *Drosophila simulans*. *PLoS Genetics 7*(2), e1001302.

Thornton, K.R. (2014). A C++ template library for efficient forward-time population genetic simulation of large populations. *Genetics 198*(1), 157–166.

Wakeley, J., King, L., Low, B.S., and Ramachandran, S. (2012). Gene genealogies within a fixed pedigree, and the robustness of Kingman's coalescent. *Genetics 190*(4), 1433–1445.

Tables

Table 1. Mean coalescence time, in generations, for a simple neutral model (Example 1; see text for model description). The mean, standard deviation (SD), standard error of the mean (SEM), and relative standard error (SE%) are shown across 500 replicates for each value of *L* simulated.

L	**mean**	**SD**	**SEM**	**SE%**
10^5	2708.8	1497.7	67.0	2.47%
10^6	4743.0	1672.6	74.8	1.58%
10^7	7897.6	1597.5	71.4	0.90%
10^8	10647.4	1485.5	66.4	0.62%
10^9	13310.4	1379.4	61.7	0.46%
10^{10}	15771.0	1354.3	60.6	0.38%

Figures

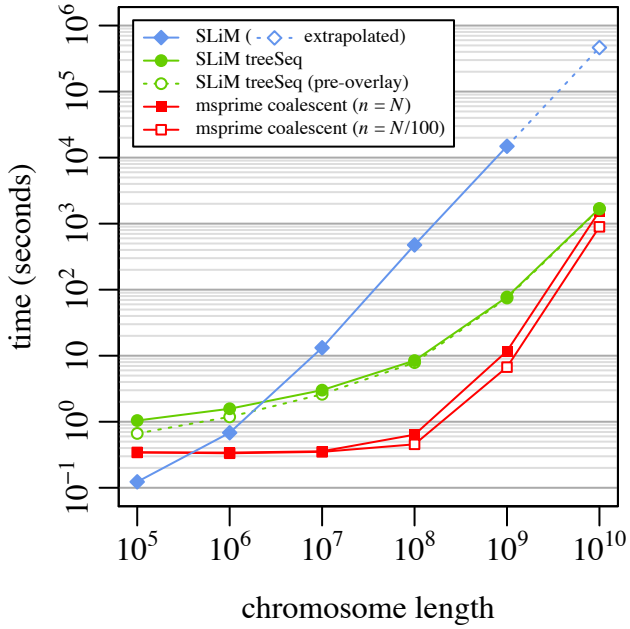

Figure 1. A speed comparison between SLiM without tree-sequence recording, SLiM with tree-sequence recording and mutation overlay, and msprime's coalescent simulation for a simple neutral model (Example 1; see text for model description). Each point represents the mean runtime across 10 replicates [NOT YET 10 RUNS FOR SLIM 1E9] using different random number seeds; bars showing standard error of the mean would be smaller than the size of the plotted points in all cases. Runs for SLiM without tree-sequence recording (filled blue diamonds) were not conducted for $L = 10^{10}$ because the memory usage was prohibitive, so a linear extrapolation is shown (hollow blue diamond). Runs for SLiM with tree-sequence recording and mutation overlay (filled green circles) are subdivided here to show the runtime for SLiM alone, prior to mutation overlay (hollow green circles), illustrating that the time for mutation overlay is negligible. The runtimes for the msprime coalescent for a full population sample of $n = N$ (filled red squares) and for a sample of size $n = N/100 = 5$ (hollow red squares) are both shown. Note that the x and y axes are both on a log scale.

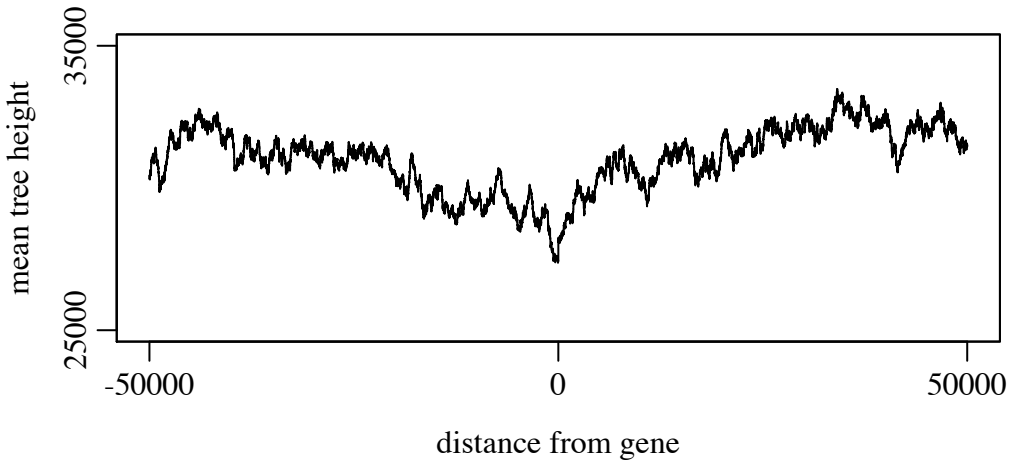

Figure 2. Mean diversity (as measured by mean tree height) as a function of distance from the nearest gene (Example 2). The center of the x-axis represents a distance of zero, immediately adjacent to a gene; moving away from the x-axis center to the left/right represents moving away from the nearest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

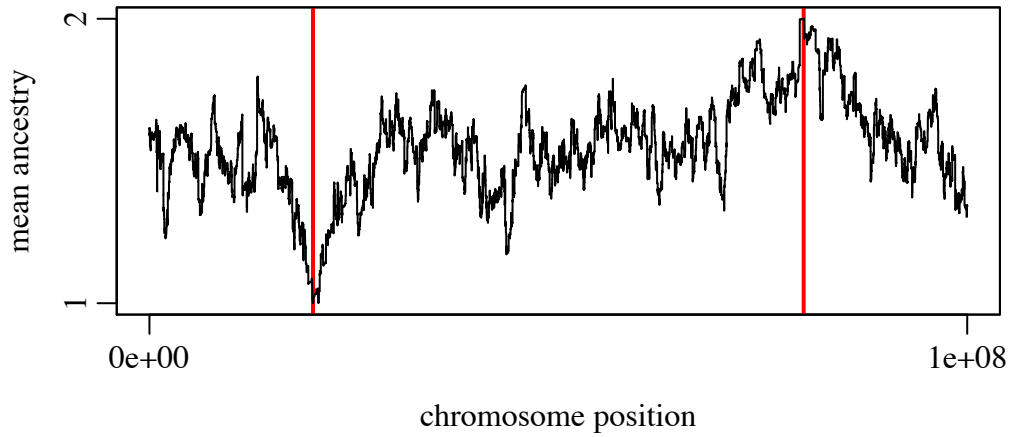

Figure 3. Mean true local ancestry at each position along the chromosome (Example 3). The red lines indicate the positions at which beneficial mutations were originally introduced into `p1` and `p2`. The beneficial mutations, which both fixed, are points where the true local ancestry is 100% `p1` or `p2`. True local ancestry regresses toward equal admixture with increasing distance from those fixed points.

[FIGURE NEEDED]

Figure 4. 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 10^8.