Simulation of bacterial populations with SLiM

Jean Cury[1], Benjamin C. Haller[2], Guillaume Achaz[3], and Flora Jay[1]

[1]*Université Paris-Saclay, Centre National de la Recherche Scientifique, Inria, Laboratoire de Recherche en Informatique UMR 8623, Orsay , France*
[2]*Department of Biological Statistics and Computational Biology, Cornell University, USA*
[3]*Université de Paris, Collège de France, Museum National d'Histoire Naturelle, Paris, France*

Abstract

Simulation of genomic data is one of the key tool in population genetics, yet, to date, there is no forward-in-time simulator of bacterial populations that is both computationally efficient and adaptable to a wide range of scenarios. Here we demonstrate how to simulate bacterial populations with SLiM, a forward-in-time simulator built for eukaryotes. SLiM gained lots of traction in the past years, and has exhaustive documentation showcasing the various scenarios that are possible to simulate. This paper focuses on a simple demographic scenario, to explore bacterial specificities and how it translates into SLiM's language. To foster the development of bacterial simulations with this recipe, we pedagogically walk the reader through the code. We also validate the simulator, by testing extensively the results of the simulations against existing simulators and theoretical expectations of some summary statistics. Finally, the combination of this protocol with the flexibility and power of SLiM enables the community to simulate efficiently bacterial populations under a wide range of evolutionary scenarios.

1 Introduction

Bacterial population genomics aims to reconstruct past evolutionary events and better understand the ongoing evolutionary dynamics operating in present-day populations. Demographic changes, selection, and migration are examples of processes whose genotypic imprints remain in present-day populations. Trying to recover these signals from ever-growing sequencing data is a major goal of population genomics. In the context of epidemiological surveillance the inference of these types of events can be useful, since pathogens are known to undergo frequent demographic changes [1]. Similarly, a better understanding of the evolutionary forces operating in pathogen populations can help to inform public health-related decisions [2]. For instance, one can assess the impact of a vaccination campaign or the efficacy of a new antibiotic on a given pathogen population [3], [4]. Apart from clinical settings, bacterial population genomics can also be useful to describe natural population diversity [5].

Simulations are essential to population genetics [6]. Many methods inferring past evolutionary events rely on simulated data. In Approximate Bayesian Computation (ABC), likely the most famous likelihood-free inference framework in our field, simulations enable to approximate the posterior distribution of the parameters of interest [7]. Other methods, based on machine learning, also require simulations to train a model to learn the mapping between the input sequence data and evolutionary processes [8]. More and more machine learning methods involve deep learning algorithms which appear to be very promising but require a large volume of simulated data to train the models [9]–[13]. Simulations are also beneficial for testing and validating population genetic methods (whether based on simulations or not), since they provide data generated by known evolutionary forces (unlike, typically, empirical sequence data). Notably, they can be used to assess the performance of methods when assumptions are violated [14], [15]. Finally, simulations can be used to forecast the impact of an environmental change on a population, or the expected response to population management [16], [17]

Despite these many applications of population genetics simulators, there are very few bacterial population genetics simulators, and the existing ones do not cover many possible scenarios. Existing bacterial simulators are coalescent-based simulators (`msPro` [18], `SimBac` [19], `FastSimBac` [20]), which means they are very fast and memory efficient, but can model only a narrow range of possible scenarios. For instance, these simulators do not allow the simulation of selective processes, and in the case of SimBac, simulation of demographic changes is not implemented. Simulation of complex selective forces together with demographic processes remains a difficult problem for coalescence-based simulators [19]. Other coalescent-based simulators that are not specific to bacteria (e.g., `ms` [21], `msprime` [22]) suffer from the same constraints, and additionally most cannot simulate bacterial recombination (similar to gene conversion). Forward simulators like `SFS_CODE` [23] can be used to produce more complex simulations with demography and selection; however, this software seems to no longer be maintained, and suffers from poor performance [24]. Yet computational efficiency is a key factor for supervised methods trained on large simulated datasets, such as ABC, machine learning, and deep learning approaches.

We present here a method for simulating bacterial populations using SLiM in a flexible and fast way. SLiM is a forward simulator [25] that is becoming more and more powerful and widely used [26]–[28]. SLiM includes a scriptable interface with its own language, Eidos, which allows simulation of a wide range of possible scenarios. The detailed instruction manual, combined with a helpful graphical user interface and SLiM's flexibility, enable users to build simulation models tailored to their research. Simulation of bacterial populations, and haploids in general, is not supported intrinsically by SLiM, because every individual has two chromosomes. But because of its scriptability, it is possible to extend SLiM into this area. In this protocol, we will show the key functions necessary to perform bacterial simulations. Following the SLiM manual's convention, we will introduce the model implementation step by step together with the related concepts. Then we will show that the simulator behaves correctly according to expected values of certain summary statistics under the Wright-Fisher model, and that the model's performance is good enough to allow numerous simulations to be run in a reasonable amount of time and memory. We believe this will open new avenues in bacterial population genetics, by allowing more researchers to use simulation-based approaches in this field and go beyond the limitations of the coalescent.

2 Methods, simulator and data

The bacterial simulator proposed here is based on SLiM, a powerful and efficient forward genetic simulator [29]. Thanks to its flexible interface and the Eidos language, we were able to adapt SLiM to the simulation of bacterial populations.

SLiM provides two types of simulations: Wright-Fisher (WF) models, and models that go beyond the Wright-Fisher framework (non-Wright-Fisher or nonWF models). The Wright-Fisher model is based on many simplifying assumptions that are often not compatible with real scenarios such as structured populations, overlapping generations, etc. [25]. However, it is mathematically simple, allowing expectations for certain quantities to be estimated. This is particularly useful to validate the created simulator against the expectations under this model. The nonWF framework, on the other hand, is more individual-based, emergent, and realistic. It allows a greater breadth of possible scenarios to be simulated, but we cannot derive expectations of the same quantities. Thus, we will provide results for the same scenario under both models to confirm that they behave similarly (according to the WF expectations).

We will present in the main text the protocol for simulating bacterial populations under the nonWF model, since it is a more powerful framework on which other users can build more complex scenarios. The corresponding annotated WF script is available in a public repository (https://github.com/jeanrjc/BacterialSlimulations), along with the nonWF script detailed below. To highlight the modeling steps that are specific to bacterial populations, we kept the underlying population history simple, with a single constant-size population and no selection, but those assumptions are trivial to relax in SLiM.

2.1 Key concepts and definitions

2.1.1 Horizontal gene transfer, recombination, and circularity

In bacteria, pieces of DNA can be exchanged between different organisms in a process called horizontal gene transfer [30]. When received, such a DNA fragment can be inserted in the host chromosome with the help of integrases, at a specific site or an arbitrary site, if the fragment is not homologous to an existing chromosomal region. Alternatively, if the incoming DNA fragment is homologous, it will integrate into the host chromosome by a mechanism similar to gene conversion in eukaryotes [31]. This latter process is the bacterial recombination mechanism that we want to implement. It differs from recombination of eukaryotes in that mutations are not exchanged between two fragments of DNA; instead, the mutations are copied from one fragment to the other. Coalescent simulators implementing gene conversion, such as `ms` [21] or `FastSimBac` [20], can be used to simulate relatively simple scenarios; however, they only model a linear chromosome, whereas most bacteria have a circular chromosome. Thus, our implementation of gene conversion provides a closer fit with reality, since we do model the bacterial chromosome as circular.

2.1.2 Burn-in

In certain situations, it is desirable to start a simulation with a population which is at mutation-drift equilibrium. Reaching this equilibrium, called the ”burn-in” of the simulation, is expected to take 5Ne generations, on average, for haploid populations. Because the effective size of bacterial populations is usually larger than the time span of interest, reaching the equilibrium with a forward simulator would require spending more than 80% of the overall simulation time in the burn-in phase – and even then, there is no guarantee that the equilibrium will be reached. To solve this issue, faster backward-in-time simulators can be used to simulate a population at equilibrium that serves to initiate the forward simulation. In the WF model, we have to simulate the entire population backward in time, and load the generated diversity into SLiM to initialize our simulation. Because we simulate the entire population, it is not possible to use gene conversion at a significant rate, otherwise `ms` crashes, thus there is no recombination in burn-in. With the nonWF model, we combine the tree sequence recording feature along with the recapitation feature of `msprime` [25]. After a forward simulation, there is often no single common ancestor for the population at the end of the simulation; in other words, the tree of the underlying population has not yet coalesced. The recapitation process will simulate, backward in time, the addition of ancestral branches to produce coalescence. Because `msprime` does not implement yet gene conversion, we cannot use bacterial recombination in burn-in. The recapitation process will be detailed for bacterial populations. Currently, it is not possible to use the tree recording feature with the WF model, because it cannot record to the tree sequence the fact that HGT is happening. As of now, it is not possible to have gene conversion during burn-in, we analyse its impact on simulation in the results section. It is important to recall that a simulation does not always start with a burn-in phase, in which case this part is not needed.

2.1.3 Simulation rescaling

Forward simulators remain computationally intensive, and bacterial populations can be very large. The effective population size of most bacterial species is on the order of $10^8 - 10^9$ [32]. Depending on the task one wants to address, many thousands or even millions of simulations may be required. One way to reduce the computational time is to parallelize the simulations on a cluster, but it can still remain costly. Another way is to rescale the model parameters such that $\theta = 2 \times Ne \times \mu$ and related quantities remain constant. For instance, we can decrease the size of the population by a factor of 10 while increasing the mutation and recombination rates by the same factor. The choice of the rescaling factor is at the discretion of the user, but one should keep in mind that excessive rescaling might lead to spurious results [33]. For instance, rescaling increases the rate of double mutation at a site, although it should remain rare [34]. Also, if the simulation involves a bottleneck, the user should make sure that the number of individuals remaining in the population after the bottleneck is not so small as to cause artifacts. With 1000 individuals, a bottleneck that reduces the population size by a factor 10 would lead to very different results if we were to rescale the model down to only 10 individuals before the bottleneck! The rescaling factor must also be applied to the duration of the simulation (and the duration of different events that might occur), so that the effects of drift remains similar. Thus, rescaled simulations not only run faster per generation (because there are fewer individuals to process), but also for a smaller number of generations. In the results section, we will show the effect of the rescaling factor on two summary statistics, along with the increase in the speed of the model. Because there are many complexities involved in rescaling, we recommend choosing this factor with great care, and cross-validating the results of downstream analyses, by doing a small number of runs that are unscaled (or less rescaled, at least).

2.2 Simulation protocol

2.2.1 Forward simulation

We now describe the protocol step by step. SLiM scripts are usually called from the command line (if they are not run within the SLiMgui graphical modeling environment). For this reason, we will set the values of constants that govern the model’s behavior with the `-d constant=value` command-line option. Our model script can be called as follows:

```
 1 $ slim -d "N_generations=1000" -d "Ne=1e5" -d "genomeSize=2e6" -d "Rho=1e-9" -d "tractlen=1e4"
       nonWF_bact.slim
```

Calling the SLiM script from bash

Using `-d constant=value` is convenient to start different simulations with different parameters. In the example above, we simulate 1000 generations of a population of 100 000 individuals, which have a chromosome of 2Mb, and a recombination rate of 10^{-9} recombinations per generation per base pair, with a mean recombination tract length of 10kb. Here is the beginning of the corresponding SLiM script:

```
 1 initialize()
 2 {
 3    initializeSLiMModelType("nonWF");
 4    initializeTreeSeq(); // record trees for recapitation and/or adding neutral mutations later
 5    initializeMutationRate(0); // no neutral mutations in the forward simulation
 6    initializeMutationType("m1", 1.0, "f", 0.0); // neutral (unused)
 7    initializeGenomicElementType("g1", m1, 1.0);
 8    initializeGenomicElement(g1, 0, genomeSize - 1);
 9 
10    initializeRecombinationRate(0); // In SLiM recombination is between sister chromatids
11    defineConstant("HGTrate", Rho * genomeSize); // HGT probability
12 }
```

Initialization of a bacterial simulation with SLiM

Here we initialize the simulation using the nonWF model with tree sequence recording, as explained in the SLiM manual. We set the mutation rate to zero because we will add neutral mutations later with `msprime`, after recapitation; we don't want to forward-simulate neutral mutations, for efficiency. Importantly for bacteria, the (generic) recombination process implemented in SLiM should not happen, otherwise, because individuals in SLiM are diploids, our haploid bacterial chromosomes will recombine with the empty second chromosomes. Thus, the recombination rate should always be set to zero, when simulating bacterial populations. Instead, we define another constant, `HGTrate`, that represents the probability of a given bacteria undergoing (homologous) HGT.

The population is created at the beginning of the first generation, as shown in the next snippet; other populations can be created here too:

```
13 1 early()
14 {
15    sim.addSubpop("p1", Ne);
16    sim.rescheduleScriptBlock(s1, start=N_generations, end=N_generations);
17 }
```

Creation of a population

In line 15, we add a subpopulation named `p1` of size `Ne`. The next line is not specific to bacteria, but allows us to define the end of the simulation dynamically, with a parameter (`N_generations`) that is given when calling this SLiM script with the `-d` option. This is useful when comparing different re-scaling factors, or when the endpoint of the simulation should depend on other parameters or events.

```
18 reproduction()
19 {
20    // each parental individual reproduces twice, with independent probabilities of HGT
21    parents = p1.individuals;
22 
23    for (rep in 0:1)
24    {
25 
26       if (HGTrate > 0)
27       {
28            // for all daughter cells, which ones are going to undergo a HGT?
29          is_HGT = rbinom(size(parents), 1, HGTrate);
30       }
31       else
32       {
33          is_HGT = integer(size(parents)); // vector of 0s
34       }
35       for (i in seqAlong(parents))
36       {
37          if (is_HGT[i])
38          {
39             // Pick another individual to receive a piece of DNA from
40             HGTsource = p1.sampleIndividuals(1, exclude=parents[i]).genome1;
41             // Choose which fragment
42             pos_beg = rdunif(1, 0, genomeSize - 1);
43             tractLength = rgeom(1, 1.0 / tractlen);
44             pos_end = pos_beg + tractLength - 1;
```

```

            // Prevent an edge case when both
            // pos_beg and tractLength are equal to 0

            if (pos_end == -1) {
               pos_end = 1;
            }
            else
            {
               pos_end = integerMod(pos_beg + tractLength - 1, genomeSize);
            }

            // HGT from pos_beg forward to pos_end on a circular chromosome
            if (pos_beg > pos_end)
               breaks = c(0, pos_end, pos_beg);
            else
               breaks = c(pos_beg, pos_end);
            subpop.addRecombinant(parents[i].genome1, HGTsource, breaks, NULL, NULL, NULL);
         }
         else
         {
            // no horizontal gene transfer; clonal replication
            subpop.addRecombinant(parents[i].genome1, NULL, NULL, NULL, NULL, NULL);
         }
      }
   }
   // deactivate the reproduction() callback for this generation
   self.active = 0;
}
```

Bacterial reproduction

In each generation, SLiM calls `reproduction()` callbacks for each individual and the callback handles how that focal individual reproduces and generates offspring. Since we want to reproduce the whole population in one big bang (for efficiency, mostly), we override that default behavior by setting `self.active = 0;` at the end of the callback. As a result, this callback is called only once per generation and will manage the reproduction of all individuals. We make each parent reproduce twice (`rep in 0:1`) to circumvent SLiM's constraint that individuals cannot undergo a horizontal gene transfer event in the middle of their lifespan. By creating two clonal offspring, each can be part of a horizontal gene transfer event; had we implemented a single clonal reproduction, only one of the two daughter cells (the one that is not the parent) could have undergone HGT. Later in the script the parents are removed from the population (by setting their fitness to 0), such that at each generation, a bacterium reproduces, and generates two cells. For each of the two offspring, we estimate which clones (`size(parents)`) will undergo an HGT event by drawing from a binomial distribution, with the probability of HGT defined by the constant 11. If an individual was chosen as a recipient for HGT, then the donor is picked randomly from the population (excluding the recipient). The DNA fragment that is going to be transferred is now defined by a starting position, drawn uniformly along the chromosome, and a length, whose value is drawn from a geometric distribution with mean (`tractlen`), defined by the user. Then, the `addRecombinant()` call creates a new daughter cell that is a clone of the parent, but with the recombination tract copied from the donor to the recipient. If the individual was not an HGT recipient, it is simply defined as a clone of its parent. Finally, as explained above, we deactivate this callback for the rest of the generation since it has just reproduced every parent.

```
early()
{
   inds = p1.individuals;
   ages = inds.age;

   // kill off parental individuals; biologically they don't even exist,
   // since they split by mitosis to generate their offspring
   inds[ages > 0].fitnessScaling = 0.0;

   // density-dependent population regulation on juveniles, toward Ne
   juvenileCount = sum(ages == 0);
   inds[ages == 0].fitnessScaling = Ne / juvenileCount;
}
```

Regulating the population size

As we saw earlier, we had to clone each individual (parent) twice, to produce two new individuals (daughter cells/juveniles). To keep the simulation realistic we now remove the parents by setting their fitness to 0. In order to simulate a demographic scenario of constant population size, and because we are under the nonWF model where the size of the population is an emergent property, and not a parameter as in WF models, we rescale the fitness of all juveniles so that the average number of individuals at each generation remains `Ne`. Before the next generation, SLiM will kill individuals based on their absolute fitness, which acts as a survival probability. Thus, at the start of the next generation we will have, on average, Ne individuals (with some stochastic fluctuation around that average).

```
s1 10000 late()
{
   sim.treeSeqOutput("mySimulation.trees");
   sim.simulationFinished();
}
```

Ending the simulation

This script block, named `s1`, was rescheduled by `rescheduleScriptBlock()` in line 16, but a scheduled time for the block to execute – here 10 000 – has to be specified even though it will be overridden with `N_generations`. This is mandatory, otherwise this block will run in each generation. The value just needs to be high enough to avoid unintended execution of the block before it gets rescheduled; the time at which the unscaled simulation would end is typically a good choice, since it will never be too early. When the simulation is over, we output the tree sequence to a `.trees` file that we can work with in Python. In the next part we will show how to generate a burn-in period and genetic diversity with `msprime`.

2.2.2 Recapitating and adding neutral mutations

To obtain a matrix of neutral SNPs – to compute summary statistics, for instance – the tree sequence can be manipulated in `msprime` with the help of `pyslim`, a python interface between SLiM and `msprime`.

```
ts = pyslim.load("mySimulation.trees")
ts_recap = ts.recapitate(recombination_rate=1e-20, # Crossing over recombination set to 0.
                   Ne=Ne)

# simplify to a subset of the population that is still alive
sample_inds = np.random.choice(ts_recap.individuals_alive_at(0),
                        size=sample_size,
                        replace=False)
# get the first node of the sampled individuals to make them haploid
sample_nodes = [ts_recap.individual(i).nodes[0] for i in sample_inds]
ts_sampled_haploid = ts_recap.simplify(samples=sample_nodes)

# Add neutral mutations
ts_mutated = pyslim.SlimTreeSequence(
                        msprime.mutate(ts_sampled_haploid,
                                       rate=mutation_rate/2, # To have 2.Ne.mu and not 4.Ne.mu
                                       keep=True) # keep existing mutations
            )


# Get the matrix of SNP, individuals in rows and SNP in columns.
snp_mat = ts_mutated.genotype_matrix().T

# get positions of the SNPs
pos = np.round(ts_mutated.tables.asdict()["sites"]["position"]).astype(int)
```

Recapitation and generation of SNPs with `pyslim` and `msprime`, in Python

First we load the tree sequence with `pyslim`, which returns an tree sequence object. We then recapitate the branches that have not coalesced yet, with a very low recombination rate since currently `msprime` does not implement gene conversion yet. We show that after a small number of forward generations (relative to Ne) this has a negligible impact, notably the expected LD is recovered (Supplementary Figures 10 and 9). Since, for this protocol, we want to generate a matrix of SNPs for a sample of individuals, not for the whole population, we subsequently sample a random subset of present-day individuals. We keep only the first node of each individual, corresponding to the first chromosome in SLiM where our haploid genetic material resides. Finally, we overlay neutral mutations on the tree sequence. We have to divide the mutation rate by two to obtain the desired $\theta = 2 \times Ne \times \mu$, instead of $\theta = 4 \times Ne \times \mu$ that is expected for diploids. At the end, we get a matrix of SNPs and a vector of corresponding positions, often used as input for inference methods [9], [12], [35].

2.3 Simulations performed

To test the simulator, we ran simulations with parameters of the bacteria *Streptoccocus agalactiae* Clonal Complex 17, which is a major neonatal pathogen [36], [37]. We used a chromosome size of 2Mb, and we estimated the following parameters based upon data we found in the literature. The simulation spans 20,000 generations, which represents about 55 years of evolution for such bacteria in the wild, when using a generation time of 1 generation per day (as estimated for E. coli [38]). The mutation rate is set to $1.53.10^{-9}$ mutations per base-pair per generation [36]. The recombination rate was set equal to the mutation rate, and the mean recombination tract length was estimated as 122 kb [39]. Note that the true recombination rate for *S. agalactiae* is probably lower [40], but in order to assess the correctness of the implementation of bacterial recombination, we chose to set it equal to the mutation rate and study the effects of varying it. The effective population size of this clonal complex was estimated to be around 140 thousand individuals [36]. At the end of the simulation, we sample 20 individuals and build a matrix of SNPs, from which we compute summary statistics.

Simulations were run on Dell server rack R640 with a processor Intel Xeon Silver 4112 2.6GHz.

3 Results

We performed two sets of experiments to assess the performance and accuracy of our simulator. In a first experiment, we assessed the impact of rescaling the effective population size, *Ne*, in order to speed up the computation time. In the second experiment, we analysed the impact of varying the recombination rate and the mean recombination tract length, to better grasp their impact on the simulations. For both experiments, we monitored the running time and peak memory usage of SLiM, and assessed the quality of the simulations by comparing the site-frequency-spectrum (SFS) and the linkage disequilibrium (LD) with simulations obtained using `ms` [21] and `FastSimBac` [20], which are backward simulators implementing bacterial recombination (or gene conversion, in `ms`).

3.1 Impact of rescaling

We compared 9 different rescaling factors (RF): 1 (no rescaling), 2, 3, 4, 5, 10, 25, 50 and 100. For RF above 2, we generated 100 replicates for each RF and each SLiM model (WF, nonWF); for RF of 1, 30 replicates were used, and for RF of 2, 50 replicates. We generated 300 replicates when running `FastSimBac` (FSB) and `ms`.

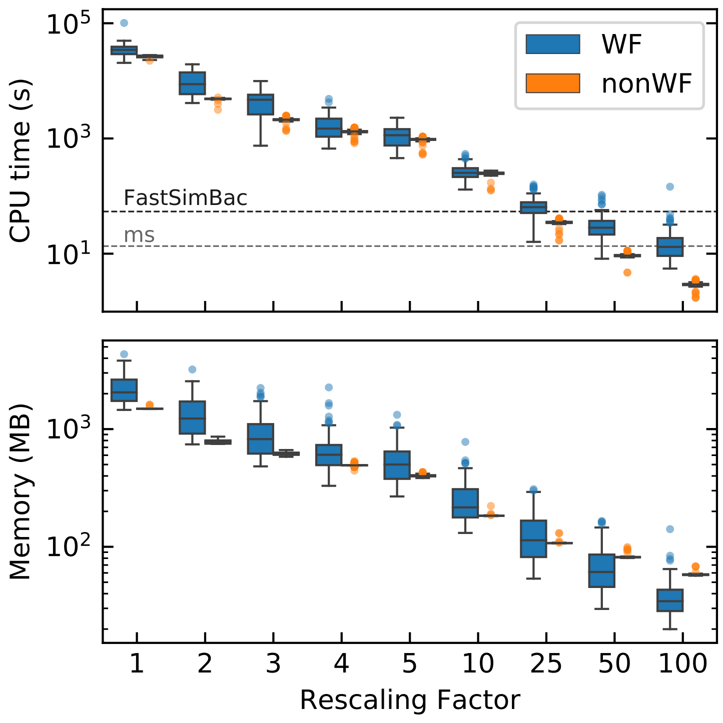

Figure 1: Distribution of the CPU time and memory peak usage for different rescaling factors. The dashed line is the average time for `FastSimBac` (~54s) and `ms` (~14s). Note the log scale on the y axis; 10^5 seconds is about 28 hours. Parameters used : chromosome size : 2Mb; $\mu = \rho = 1.53 \times 10^{-9}$; Ne = 140k; 20000 generations.

Without rescaling, the generation of a single replicate takes about a day (Figure 1). This is too long if one wants to run millions of simulations; however, it is possible to do a few such runs for other purposes, such as confirming that rescaling did not introduce a bias when implementing a new script. This might also be useful to test a method on a dataset produced without rescaling, since even minor artifacts introduced by rescaling could conceivably bias or confuse inference methods. When using a rescaling factor of 5, a simulation takes about 1 hour to run, which is practicable if one wants to run thousands of simulations on a cluster. With a factor of 25 or more, the running time is comparable to that of `FastSimBac` and `ms`, if not faster; `FastSimBac` is a bit slower than `ms`, probably because we used the additional `FastSimBac` script to create an `ms`-formatted output file. At 100 seconds or less per replicate, it is possible to generate about a million replicates in few days or a week, on a typical computing cluster (using perhaps 100 cores). The time of the burn-in period is included here, and is not a limiting factor since it is faster than the forward-simulation period by about two orders of magnitude for rescaling factors smaller than 5 (supplementary Figure 7). The memory peak usage is fairly low (up to a few gigabytes without rescaling), allowing any modern laptop to run these simulations.

Comparing WF and nonWF performance, we see that the nonWF model tends to be faster, especially at higher rescaling factors. This is due to the overhead of the burn-in step, which is slower in the WF models. Without rescaling, or at lower rescaling factors, the difference between WF and nonWF running time tends to disappear. Interestingly, the variance in time and in memory is less variable for the nonWF version, which can be useful to better predict the resources needed for large runs. It is important to note that these performance metrics depend on the parameter values used (such as the recombination rate).

We then computed the normalized SFS produced by the different rescaling factors. The SFS represents the distribution of the frequency of derived alleles. Each bin (i) is given by $i\xi_i/\hat{\theta}$, where ξ_i is the count of SNPs having i derived alleles, and $\hat{\theta}$ is an estimator of θ computed as the mean over the different bins ($1/n \sum i\xi_i$). Because $i\xi_i$ is an estimator of θ, the expected normalized SFS for a constant size neutral population under the Wright-Fisher model is a flat line centered on 1 [41], [42].

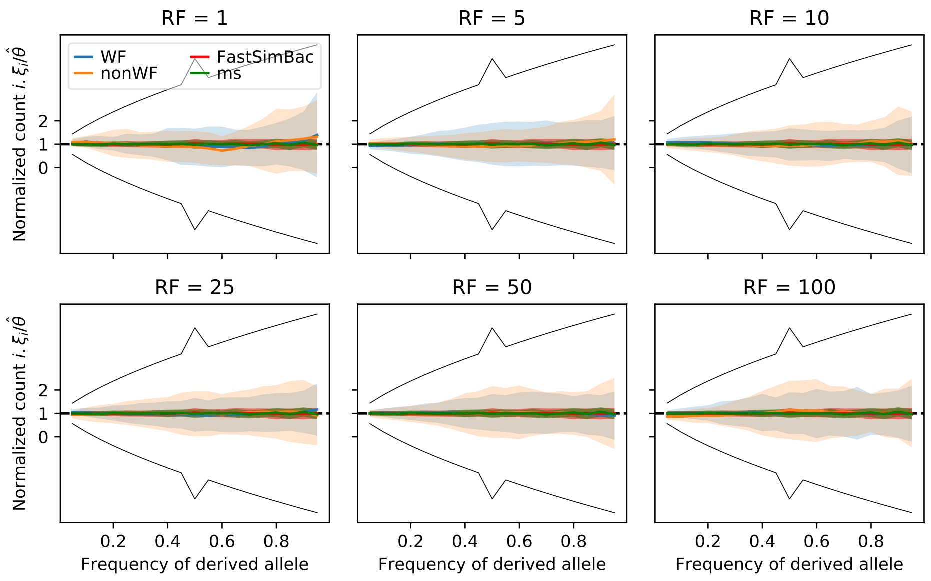

Figure 2: Normalized Site Frequency Spectrum (SFS) for different rescaling factors. The shaded area represents the standard deviation. The black lines represent the expected standard deviation. Parameters used : chromosome size : 2Mb; $\mu = \rho = 1.53 \times 10^{-9}$; Ne = 140k; 20000 generations.

Figure 2 shows the normalized SFS for 6 rescaling factors (see supplementary figure 2 for all RF) with the expected standard deviation under the Wright-Fisher model without recombination [41]. `FastSimBac` and `ms` simulations are here as a second control in addition to the theoretical expectations (horizontal line at 1).

We see that all experiments leads to the expected SFS, well within the expected standard deviation for linked loci. A lower standard deviation is expected, as recombination is known to decrease the variance of the SFS [43]. Thus, rescaling factors up to 100 with this set of parameters do not affect the SFS, which behaves correctly for WF and nonWF models.

Next, we assessed the impact of recombination on linkage disequilibrium (LD). The LD is measured by r^2, which quantifies how much correlation (or linkage) there is between two alleles with a given distance between them. We measured this correlation by subsampling pairs of SNPs, in 19 bins of increasing distances. The LD is represented as a function of the mean distance within each bin. We compared to the LD obtained with simulations from `FastSimBac` and `ms`. In figure 3 we observe that the LD for WF and nonWF are similar to those obtained with `ms` and `FastSimBac`, and does not seem to be affected by the rescaling factor. Unlike for the SFS, the expected LD and expected variation are much harder to obtain and are out of the scope of this paper. However, we know that the LD at very short distance should be close to the LD obtained in absence of recombination. In figure 3, we represented the range of LD without recombination at short distance, with the shaded gray area. More precisely, it shows the highest and lowest values of mean plus or minus standard error of the mean, respectively, of the four simulators without recombination. The full LD without recombination can be seen in supplementary figures 10 and 11. There might exist however a small difference between backward and forward simulators as the backward ones tend to produce higher LD at short distance than the forward ones. It might be due to different implementation of recombination at short distance, or to the lack of recombination in the burn-in part. Overall, we show that all types of simulations produce expected LD at short distances and converges toward the expected r^2 with free recombination of $1/n$ (dashed line) [44], [45].

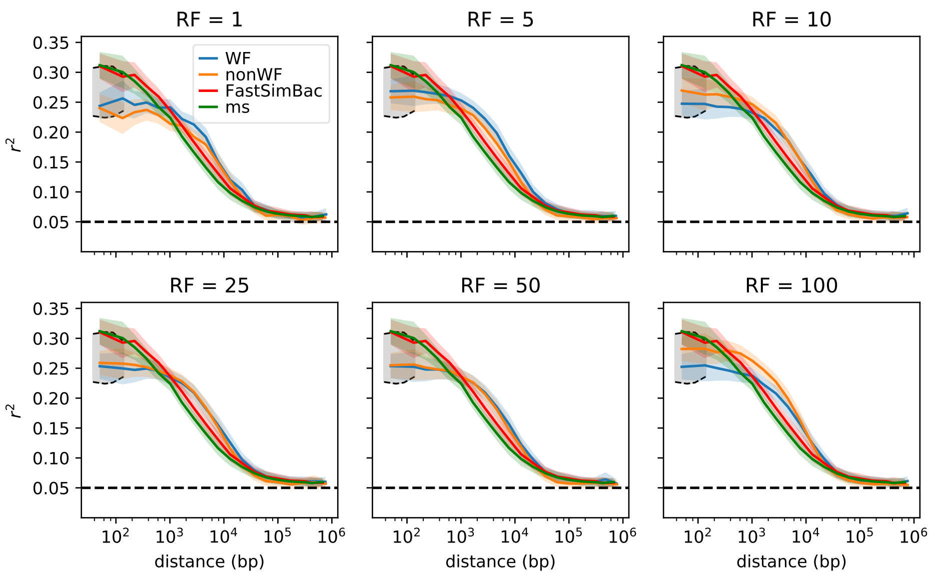

Figure 3: Linkage disequilibrium for WF and nonWF simulations and for `ms` and FSB, with different rescaling factor (RF). The horizontal dash line is the expected r^2 with free recombination when sampling 20 individuals (1/20). The colored shaded area represents the standard error of the mean, and the gray area represents the range of expected value at very short distances. Parameters used : chromosome size : 2Mb; $\mu = \rho = 1.53 \times 10^{-9}$; Ne = 140k; 20000 generations.

Overall, rescaling the simulations up to a factor of one hundred produces the expected SFS and LD, while allowing a drastic reduction in time and memory. This opens the possibility of running many simulations in a small amount of time, allowing the power and flexibility of forward simulation to be leveraged much more usefully in bacterial population genomics.

3.2 Impact of recombination

In this section we assess the impact of recombination with the same set of parameters used previously, with a rescaling factor of 25 across all forthcoming runs. We compare simulations under three recombination rates ($\rho/10$, ρ and 10ρ, where $\rho = 1.53 \times 10^{-9}$) and three mean tract lengths ($\lambda/100$, $\lambda/10$, λ, where $\lambda = 122\ kb$). First looking at performance, the run time increases 18-fold when the recombination rate increases by a factor of 100 for the WF simulations, while for the same recombination increase, the run time increases by only about 3-fold for the nonWF simulations (Figure 4 top). However, higher recombination rates require more memory when using the nonWF model, but still an affordable amount for a modern laptop (less than 1 GB, Figure 4 bottom) with the rescaling factor used in this experiment.

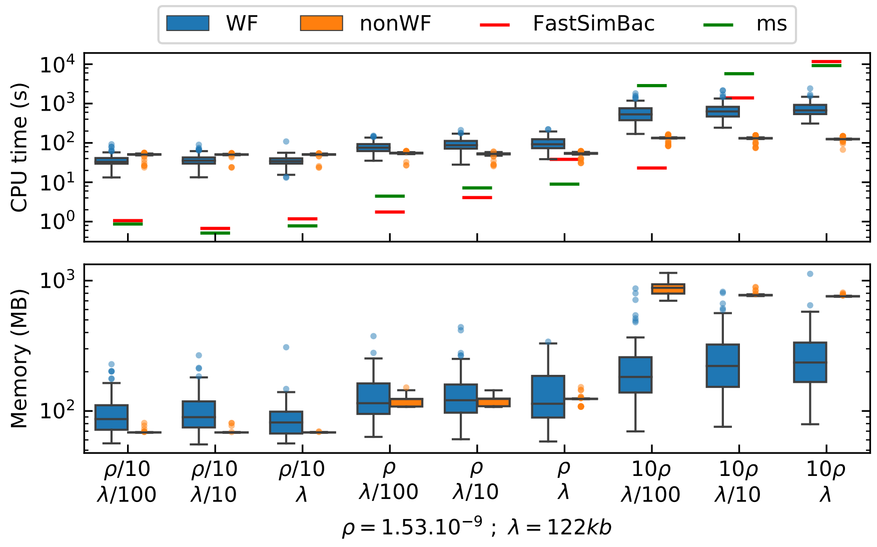

Figure 4: Distribution of the CPU time and memory peak usage for different recombination rates (ρ) and mean recombination tract lengths (λ). Parameters used : chromosome size : 2Mb; $\mu = 1.53 \times 10^{-9}$; Ne = 140k; 30000 generations; RF=25.

The recombination rate thus has an important impact on the run time of the WF simulations, but has much less impact on the nonWF simulations. The size of the recombination tract does not seem to significantly affect either the run time or the memory usage. As expected, coalescent simulators are very fast with a low recombination rate, but struggle with a higher recombination rate [20]. It takes them up to 10 thousand times longer to run when increasing the recombination rate by a factor of 100. Because of this, the simulations with `ms` and `FastSimBac` with 10ρ and λ were too slow, so we could only run 6 and 7 replicates, respectively, instead of a hundred.

As in the previous experiment, we analysed the behaviour of our simulations with respect to the normalized SFS and the LD. In figure 5, we see that the SFS is distributed as expected (flat line centered at 1), independently of the simulator or type of simulation. Interestingly, we observed two expected theoretical results. The standard deviation of the simulated SFS at low recombination matches expectation [41], and the variance decreases as the recombination rate increases [43]. We see that for a given recombination rate, decreasing the recombination tract length has a similar effect as decreasing the recombination rate for a given tract length (moving between figure panels leftward is similar to moving between figure panels upward).

In figure 6 the decay of LD with distance is similar when comparing all four types of simulations. We observe the same pattern between coalescent and SLiM simulations as we saw earlier, but only for a subset of the parameters. At low recombination rate, we recover the clonal frame, corresponding to the fact that bacterial recombination involves small patches of homologous DNA, rather than long stretches [46]. It means that positions on either side of that patch will stay linked, and this explains the space between the line of the expected LD with free recombination and the LD curve at high distance, which is expected in bacteria [31]. A higher recombination rate or a longer recombination tract length tends to approximate the expected LD of an organism with recombination by crossing over rather than HGT.

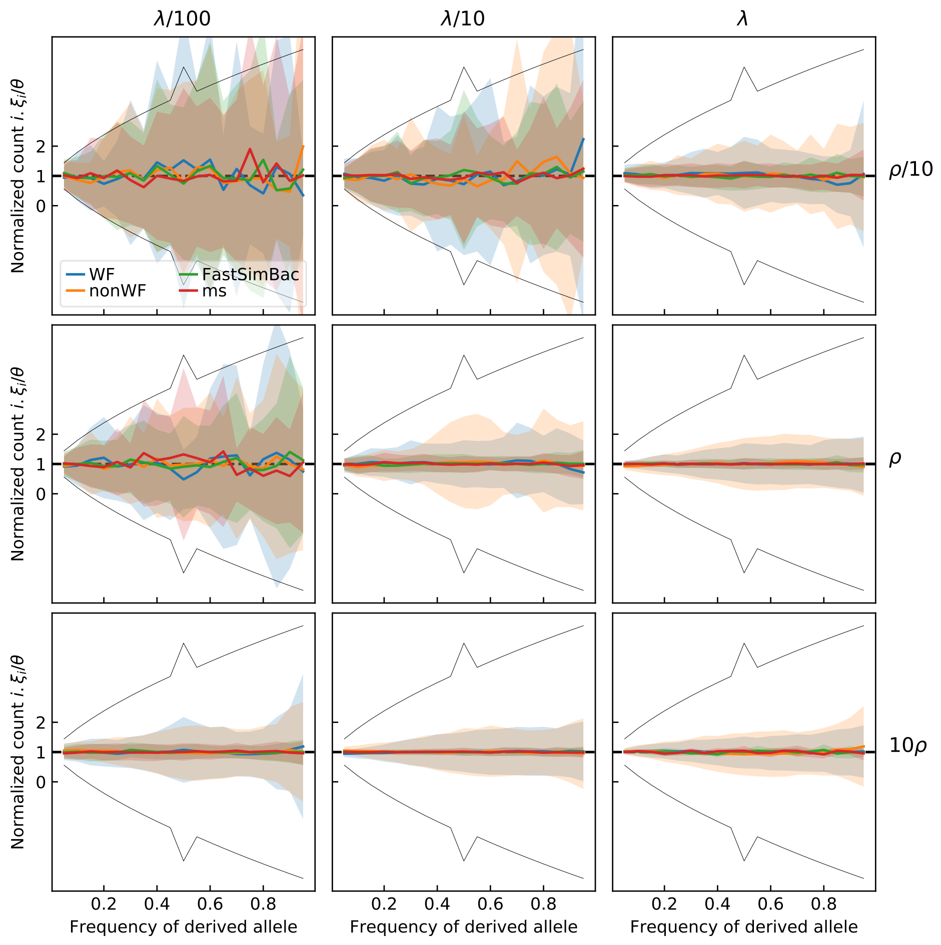

Figure 5: Normalized Site Frequency Spectrum (SFS) for different recombination rates (ρ) and tract lengths (λ). The colored shaded area represent the standard deviation. The black lines represent the expected standard deviation. Parameters used : chromosome size = 2Mb; $\mu = \rho = 1.53 \times 10^{-9}$; $\lambda = 122kb$; Ne = 140k; 30000 generations; RF=25

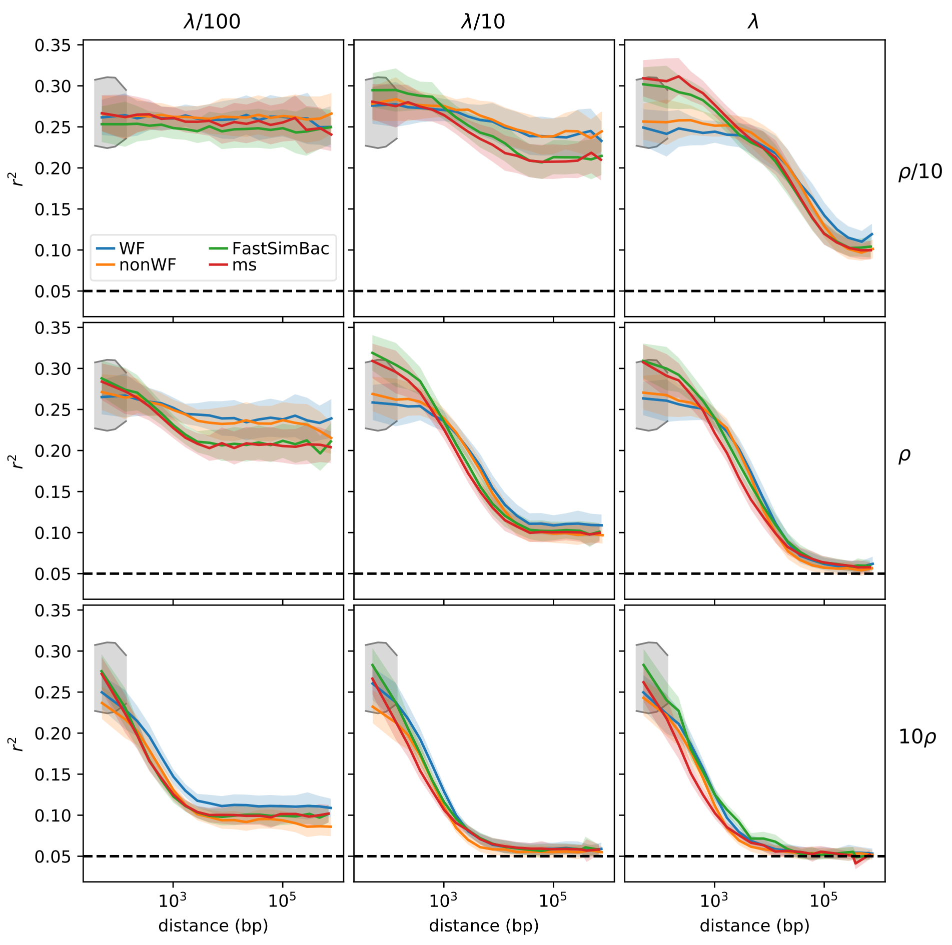

Figure 6: Linkage disequilibrium for WF and nonWF simulations with various recombination rates (ρ) and tract lengths (λ). The colored shaded area represents the standard error of the mean, and the gray area represents the range of expected value at very short distances. The horizontal black dash line is the expected r^2 with free recombination when sampling 20 individuals (1/20). Parameters used : chromosome size = 2Mb; $\mu = \rho = 1.53 \times 10^{-9}$; $\lambda = 122kb$; Ne = 140k; 30000 generations; RF=25

Overall, changing the recombination rate and mean recombination tract length produced the expected statistical results. Even for the highest recombination rate, the run time and memory requirements are still low enough to allow many simulations to be run (Figure 4), and if necessary, one might increase the rescaling factor (with proper validation and testing). Interestingly, with a high recombination rate the rescaled SLiM simulations were much faster than coalescent simulations. Finally, the nonWF model seems to have a more predictable run time and memory footprint, which might be beneficial when computing resources are scarce.

4 Discussion

We presented here a step-by-step protocol for performing simulations of a simple bacterial population. Although SLiM is not focused on bacteria, the simulations were shown to behave correctly, and could run in a reasonable amount of time. The models presented here were simple to draw attention of the specific details involving in simulating bacterial populations, but all of the model variation discussed in the SLiM manual – complex demography and population structure, selection, and so forth – could easily be added to this foundational model. This simplified approach also allowed us to compare the accuracy of our implementation to theoretical expectations, and to other simulators for which substantially more complex scenarios would not have been possible.

These simulations were made both within the Wright-Fisher framework and within the more individual-based nonWF framework, to showcase these two possibilities for the user. The recipes for these WF and nonWF models are freely available on GitHub (https://github.com/jeanrjc/BacterialSlimulations), where we encourage everyone to propose their recipes for more complex scenarios. It might be worth mentioning why one would choose a WF or nonWF model in SLiM, since this fundamental choice will guide much of the model development that follows. The WF model is simpler in many ways: it involves more simplifying assumptions and less individual-level behavior. For example, population size in the WF model is automatically maintained at a set level, whereas the nonWF model requires you to write script that regulates the population size via mechanisms such as density-dependence or – appropriately for pathogens, perhaps – host mortality. Similarly, reproduction in the WF model is automatic, based upon fitness; high-fitness individuals reproduce more than low-fitness individuals, a fact that SLiM automatically enforces, whereas in nonWF models fitness typically influences mortality, not fecundity, and reproduction is explicitly scripted to allow for greater individual-level variation in the modes and mechanisms of offspring generation. Writing a nonWF model is therefore a bit more complex and technical, and requires more details to be spelled out explicitly. Normally, nonWF models are slower, but here the slower implementation of the burn-in in WF models due to the impossibility to use the tree-recording feature with bacterial recombination meant that the WF models were slower. This is, in part, why we emphasized the nonWF model here; in this context, it really provides both greater power and flexibility, and better performance. However, the WF model remains simpler, conceptually and in its implementation; and if one wants fitness to affect fecundity rather than mortality it can be the more natural choice.

Currently, the only drawback of this simulator concerns the burn-in step which lacks recombination, due to technical limitation for `ms` and due to lack of gene conversion in `msprime` yet. The implementation of gene conversion in `msprime` is an on-going work and may be available soon and will greatly improve the overall model. This lack of recombination in burn-in leads to a lack of LD when running a forward simulation not long enough. We see that starting from 20 000 forward generations (about Ne/7 generations), LD and SFS matches that of `ms` and `FastSimBac` (supplementary Figure 9 and Figure 10). If one wants to run a very short simulation and require burn-in, it might be worth running at least Ne/7 generations, as long as `msprime` does not implement gene conversion. The higher variance of the SFS observed in the experiments of the SLiM simulations compared to `ms` and `FastSimBac` might be explained by this lack of recombination in burn-in since recombination decreases SFS's variance [43]. Importantly, burn-in is not mandatory as the simulation can involve only the forward part. However, when needed, a few steps of forward generations are enough to recover the correct LD.

We hope that our work here will stimulate a wave of development of simulation-based models for bacterial population genetics. We believe that this paper, combined with the hundred-plus models presented in SLiM's extensive documentation, will allow anyone to create new scenarios for bacterial populations seamlessly. It is possible to simulate evolution in continuous space (such as in a Petri dish), to model nucleotides explicitly (including the use of FASTA and VCF files), to model selection based on external environmental factors such as the presence of antibiotics (and selection for resistance genes), and even to model within-host evolution using a single subpopulations for each host while modeling between-host transmission and infectivity dynamics; with the scriptability of SLiM almost anything is possible. We look forward to seeing the diverse research questions that the bacterial genomics community will explore with SLiM.

Acknowledgements

We thank Peter Ralph, Eduardo Rocha and Philippe Glaser for fruitful discussions. We thank DIM One Health 2017 (number RPH17094JJP) and Human Frontier Science Project (number RGY0075/2019) for funding.

References

[1] J. B. H. Martiny, B. J. Bohannan, J. H. Brown, R. K. Colwell, J. A. Fuhrman, J. L. Green, M. C. Horner-Devine, M. Kane, J. A. Krumins, C. R. Kuske, P. J. Morin, S. Naeem, L. Øvreås, A.-L. Reysenbach, V. H. Smith, and J. T. Staley, "Microbial biogeography: Putting microorganisms on the map," en, *Nature Reviews Microbiology*, vol. 4, no. 2, pp. 102–112, Feb. 2006, ISSN: 1740-1526, 1740-1534. DOI: `10.1038/nrmicro1341`.

[2] Y. H. Grad and M. Lipsitch, "Epidemiologic data and pathogen genome sequences: A powerful synergy for public health," en, *Genome Biology*, vol. 15, no. 11, p. 538, Nov. 2014, ISSN: 1474-760X. DOI: `10.1186/s13059-014-0538-4`.

[3] N. J. Croucher, J. A. Finkelstein, S. I. Pelton, P. K. Mitchell, G. M. Lee, J. Parkhill, S. D. Bentley, W. P. Hanage, and M. Lipsitch, "Population genomics of post-vaccine changes in pneumococcal epidemiology," *Nature Publishing Group*, vol. 45, no. 6, 2013. DOI: `10.1038/ng.2625`.

[4] N. J. Croucher, C. Chewapreecha, W. P. Hanage, S. R. Harris, L. Mcgee, M. Van Der Linden, J.-H. Song, K. S. Ko, H. De Lencastre, C. Turner, F. Yang, R. Sá-Leã O, B. Beall, K. P. Klugman, J. Parkhill, P. Turner, and S. D. Bentley, "Evidence for Soft Selective Sweeps in the Evolution of Pneumococcal Multidrug Resistance and Vaccine Escape," *Genome Biol. Evol*, vol. 6, no. 7, pp. 1589–1602, 2014. DOI: `10.1093/gbe/evu120`.

[5] D. A. Robinson, D. Falush, and E. J. Feil, *Bacterial Population Genetics in Infectious Disease*, en, Wiley-Blackwell. 2010, ISBN: 978-0-470-42474-2.

[6] S. Hoban, "An overview of the utility of population simulation software in molecular ecology," en, *Molecular Ecology*, vol. 23, no. 10, pp. 2383–2401, May 2014, ISSN: 09621083. DOI: `10.1111/mec.12741`.

[7] K. Csilléry, M. G. Blum, O. E. Gaggiotti, and O. François, "Approximate Bayesian Computation (ABC) in practice," en, *Trends in Ecology & Evolution*, vol. 25, no. 7, pp. 410–418, Jul. 2010, ISSN: 01695347. DOI: `10.1016/j.tree.2010.04.001`.

[8] D. R. Schrider and A. D. Kern, "Supervised Machine Learning for Population Genetics: A New Paradigm," *Trends in Genetics*, vol. 34, no. 4, pp. 301–312, Apr. 2018, ISSN: 0168-9525. DOI: `10.1016/j.tig.2017.12.005`.

[9] S. Sheehan and Y. S. Song, "Deep Learning for Population Genetic Inference," *PLOS Computational Biology*, vol. 12, no. 3, K. Chen, Ed., e1004845–e1004845, Mar. 2016. DOI: `10.1371/journal.pcbi.1004845`.

[10] A. D. Kern and D. R. Schrider, "diploS/HIC: An Updated Approach to Classifying Selective Sweeps," en, *G3: Genes, Genomes, Genetics*, g3.200262.2018, Apr. 2018, ISSN: 2160-1836. DOI: `10.1534/g3.118.200262`.

[11] L. Flagel, Y. Brandvain, and D. R. Schrider, "The Unreasonable Effectiveness of Convolutional Neural Networks in Population Genetic Inference," en, *Molecular Biology and Evolution*, vol. 36, no. 2, pp. 220–238, Feb. 2019, ISSN: 0737-4038. DOI: `10.1093/molbev/msy224`.

[12] T. Sanchez, J. Cury, G. Charpiat, and F. Jay, "Deep learning for population size history inference: Design, comparison and combination with approximate Bayesian computation," en, *Molecular Ecology Resources*, pp. 1755–0998.13224, Jul. 2020, ISSN: 1755-098X, 1755-0998. DOI: `10.1111/1755-0998.13224`.

[13] C. Battey, P. L. Ralph, and A. D. Kern, "Predicting geographic location from genetic variation with deep neural networks," en, *eLife*, vol. 9, e54507, Jun. 2020, ISSN: 2050-084X. DOI: `10.7554/eLife.54507`.

[14] M. Lapierre, C. Blin, A. Lambert, G. Achaz, and E. P. C. Rocha, "The Impact of Selection, Gene Conversion, and Biased Sampling on the Assessment of Microbial Demography," *Molecular Biology and Evolution*, vol. 33, no. 7, pp. 1711–1725, Jul. 2016. DOI: `10.1093/molbev/msw048`.

[15] L. Chikhi, V. C. Sousa, P. Luisi, B. Goossens, and M. A. Beaumont, "The Confounding Effects of Population Structure, Genetic Diversity and the Sampling Scheme on the Detection and Quantification of Population Size Changes," en, *Genetics*, vol. 186, no. 3, pp. 983–995, Nov. 2010, ISSN: 0016-6731, 1943-2631. DOI: `10.1534/genetics.110.118661`.

[16] F. Jay, S. Manel, N. Alvarez, E. Y. Durand, W. Thuiller, R. Holderegger, P. Taberlet, and O. François, "Forecasting changes in population genetic structure of alpine plants in response to global warming," en, *Molecular Ecology*, vol. 21, no. 10, pp. 2354–2368, May 2012, ISSN: 09621083. DOI: `10.1111/j.1365-294X.2012.05541.x`.

[17] M. Bruford, M. Ancrenaz, L. Chikhi, I. Lackmann-Ancrenaz, M. Andau, L. Ambu, and B. Goossens, "Projecting genetic diversity and population viability for the fragmented orangutan population in the Kinabatangan floodplain, Sabah, Malaysia," en, *Endangered Species Research*, vol. 12, no. 3, pp. 249–261, Oct. 2010, ISSN: 1863-5407, 1613-4796. DOI: `10.3354/esr00295`.

[18] T. Akita, S. Takuno, and H. Innan, "Coalescent framework for prokaryotes undergoing interspecific homologous recombination," en, *Heredity*, Jan. 2018, ISSN: 0018-067X, 1365-2540. DOI: `10.1038/s41437-017-0034-1`.

[19] T. Brown, X. Didelot, D. J. Wilson, and N. D. Maio, "SimBac: Simulation of whole bacterial genomes with homologous recombination," en, *Microbial Genomics*, vol. 2, no. 1, Jan. 2016, ISSN: 2057-5858, 2057-5858. DOI: 10.1099/mgen.0.000044.

[20] N. De Maio and D. J. Wilson, "The Bacterial Sequential Markov Coalescent," en, *Genetics*, vol. 206, no. 1, pp. 333–343, May 2017, ISSN: 0016-6731, 1943-2631. DOI: 10.1534/genetics.116.198796.

[21] R. R. Hudson, "Ms a program for generating samples under neutral models," 2004.

[22] J. Kelleher, A. M. Etheridge, and G. McVean, "Efficient Coalescent Simulation and Genealogical Analysis for Large Sample Sizes," en, *PLOS Computational Biology*, vol. 12, no. 5, e1004842, May 2016, ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1004842.

[23] R. D. Hernandez, "A flexible forward simulator for populations subject to selection and demography," en, *Bioinformatics*, vol. 24, no. 23, pp. 2786–2787, Dec. 2008, ISSN: 1367-4803, 1460-2059. DOI: 10.1093/bioinformatics/btn522.

[24] B. C. Haller and P. W. Messer, "SLiM 2: Flexible, Interactive Forward Genetic Simulations," *Molecular Biology and Evolution*, vol. 34, no. 1, pp. 230–240, Jan. 2017, ISSN: 0737-4038. DOI: 10.1093/molbev/msw211.

[25] ——, "SLiM 3: Forward Genetic Simulations Beyond the Wright–Fisher Model," en, *Molecular Biology and Evolution*, p. 6, Jan. 2019. DOI: 10.1093/molbev/msy228.

[26] A. M. Sackman, R. B. Harris, and J. D. Jensen, "Inferring Demography and Selection in Organisms Characterized by Skewed Offspring Distributions," en, *Genetics*, vol. 211, pp. 1019–1028, 2019.

[27] G. S. Bradburd and P. L. Ralph, "Spatial Population Genetics: It's About Time," en, *Annual Review of Ecology, Evolution, and Systematics*, vol. 50, no. 1, pp. 427–449, Nov. 2019, ISSN: 1543-592X, 1545-2069. DOI: 10.1146/annurev-ecolsys-110316-022659.

[28] J. Kelleher, Y. Wong, A. W. Wohns, C. Fadil, P. K. Albers, and G. McVean, "Inferring whole-genome histories in large population datasets," en, *Nature Genetics*, vol. 51, no. 9, pp. 1330–1338, Sep. 2019, ISSN: 1061-4036, 1546-1718. DOI: 10.1038/s41588-019-0483-y.

[29] B. C. Haller and P. W. Messer, "Evolutionary Modeling in SLiM 3 for Beginners," en, *Molecular Biology and Evolution*, vol. 36, no. 5, R. Hernandez, Ed., pp. 1101–1109, May 2019, ISSN: 0737-4038, 1537-1719. DOI: 10.1093/molbev/msy237.

[30] H. Ochman, J. G. Lawrence, and E. A. Groisman, "Lateral gene transfer and the nature of bacterial innovation.," *Nature*, vol. 405, no. 6784, pp. 299–304, May 2000, ISSN: 0028-0836. DOI: 10.1038/35012500.

[31] E. P. C. Rocha, "Neutral Theory, Microbial Practice: Challenges in Bacterial Population Genetics," en, *Molecular Biology and Evolution*, vol. 35, no. 6, pp. 1338–1347, Jun. 2018, ISSN: 0737-4038. DOI: 10.1093/molbev/msy078.

[32] L.-M. Bobay and H. Ochman, "Factors driving effective population size and pan-genome evolution in bacteria," en, *BMC Evolutionary Biology*, vol. 18, no. 1, p. 153, Dec. 2018, ISSN: 1471-2148. DOI: 10.1186/s12862-018-1272-4.

[33] B. C. Haller, *SLiM3 Manual*, 2020.

[34] C. J. Hoggart, M. Chadeau-Hyam, T. G. Clark, R. Lampariello, J. C. Whittaker, M. De Iorio, and D. J. Balding, "Sequence-Level Population Simulations Over Large Genomic Regions," en, *Genetics*, vol. 177, no. 3, pp. 1725–1731, Nov. 2007, ISSN: 0016-6731. DOI: 10.1534/genetics.106.069088.

[35] F. Jay, S. Boitard, and F. Austerlitz, "An ABC Method for Whole-Genome Sequence Data: Inferring Paleolithic and Neolithic Human Expansions," en, *Molecular Biology and Evolution*, vol. 36, no. 7, pp. 1565–1579, Jul. 2019, ISSN: 0737-4038. DOI: 10.1093/molbev/msz038.

[36] V. Da Cunha, M. R. Davies, P.-E. Douarre, I. Rosinski-Chupin, I. Margarit, S. Spinali, T. Perkins, P. Lechat, N. Dmytruk, E. Sauvage, L. Ma, B. Romi, M. Tichit, M.-J. Lopez-Sanchez, S. Descorps-Declere, E. Souche, C. Buchrieser, P. Trieu-Cuot, I. Moszer, D. Clermont, D. Maione, C. Bouchier, D. J. Mcmillan, J. Parkhill, J. L. Telford, G. Dougan, M. J. Walker, T. D. Consortium, M. T. G. Holden, C. Poyart, and P. Glaser, "Streptococcus agalactiae clones infecting humans were selected and fixed through the extensive use of tetracycline," *Nature communications*, vol. 5, no. 4544, 2014. DOI: 10.1038/ncomms5544.

[37] S. Bellais, A. Six, A. Fouet, M. Longo, N. Dmytruk, P. Glaser, P. Trieu-Cuot, and C. Poyart, "Capsular Switching in Group B Streptococcus CC17 Hypervirulent Clone: A Future Challenge for Polysaccharide Vaccine Development," en, *Journal of Infectious Diseases*, vol. 206, no. 11, pp. 1745–1752, Dec. 2012, ISSN: 0022-1899, 1537-6613. DOI: `10.1093/infdis/jis605`.

[38] M. A. Savageau, "Escherichia coli habitats, cell types, and molecular mechanisms of gene control," *The american naturalist*, vol. 122, no. 6, pp. 732–744, 1983.

[39] M. Brochet, C. Rusniok, E. Couvé, S. Dramsi, C. Poyart, P. Trieu-Cuot, F. Kunst, and P. Glaser, "Shaping a bacterial genome by large chromosomal replacements, the evolutionary history of Streptococcus agalactiae," *Proceedings of the National Academy of Sciences*, vol. 105, no. 41, pp. 15 961–15 966, 2008.

[40] T. Lefébure and M. J. Stanhope, "Evolution of the core and pan-genome of Streptococcus: Positive selection, recombination, and genome composition," *Genome Biology*, vol. 8, R71, May 2007, ISSN: 1474-760X. DOI: `10.1186/gb-2007-8-5-r71`.

[41] Y.-X. Fu, "Statistical Properties of Segregating Sites," *Theoretical Population Biology*, vol. 48, pp. 172–197, 1995.

[42] G. Achaz, "Frequency Spectrum Neutrality Tests: One for All and All for One," en, *Genetics*, vol. 183, no. 1, pp. 249–258, Sep. 2009, ISSN: 0016-6731, 1943-2631. DOI: `10.1534/genetics.109.104042`.

[43] J. D. Wall, "Recombination and the power of statistical tests of neutrality," en, *Genetical Research*, vol. 74, no. 1, pp. 65–79, Aug. 1999, ISSN: 00166723. DOI: `10.1017/S0016672399003870`.

[44] S. Takuno, T. Kado, R. P. Sugino, L. Nakhleh, and H. Innan, "Population Genomics in Bacteria: A Case Study of Staphylococcus aureus," en, *Molecular Biology and Evolution*, vol. 29, no. 2, pp. 797–809, Feb. 2012, ISSN: 0737-4038. DOI: `10.1093/molbev/msr249`.

[45] R. S. Waples, "A bias correction for estimates of effective population size based on linkage disequilibrium at unlinked gene loci*," en, *Conservation Genetics*, vol. 7, no. 2, pp. 167–184, Apr. 2006, ISSN: 1566-0621, 1572-9737. DOI: `10.1007/s10592-005-9100-y`.

[46] R. Milkman and M. M. Bridges, "Molecular evolution of the Escherichia coli chromosome. III. Clonal frames.," *Genetics*, vol. 126, no. 3, pp. 505–517, 1990.

Supplementary Figures

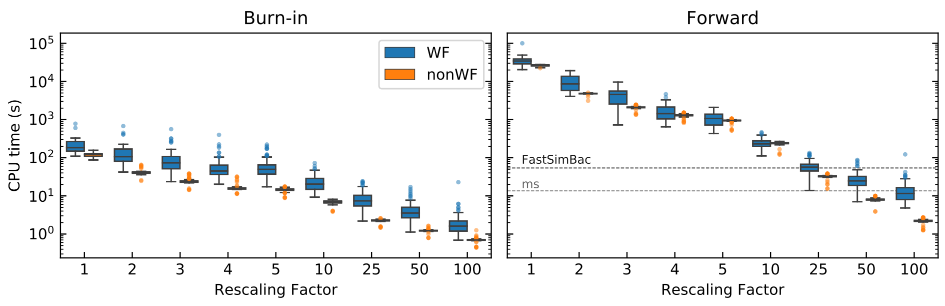

Figure 7: Computing time as in Figure 1, but split between burn-in and forward parts

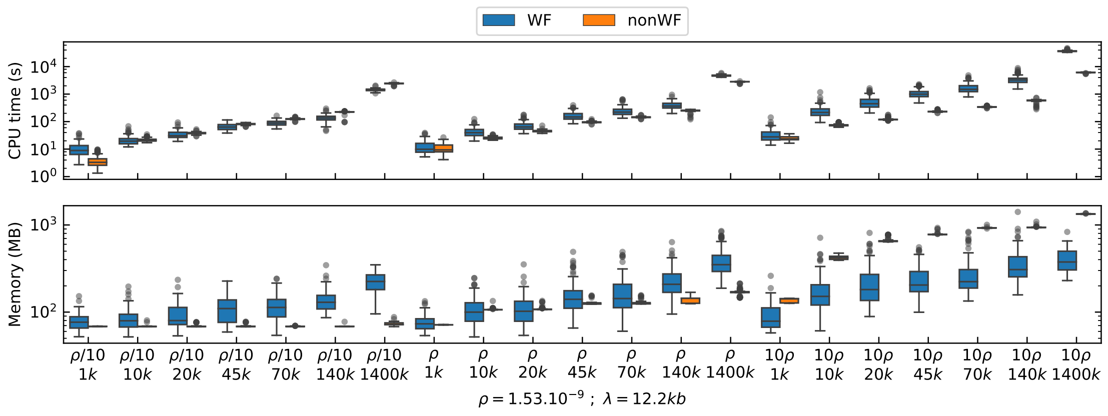

Figure 8: Computing time for different number of forward generations with three different recombination rates. Ne=140 000, λ = 12200bp and RF=25

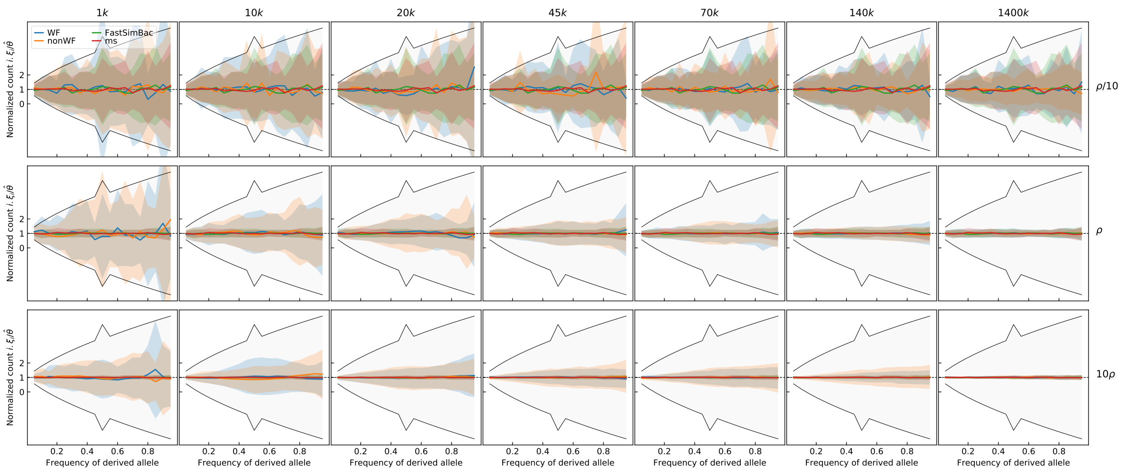

Figure 9: Normalized Site Frequency Spectrum (SFS) for different number of forward generations and recombination rates. The shaded area represents the standard deviation. The black lines represent the expected standard deviation. Ne=140 000, λ = 12200bp and RF=25

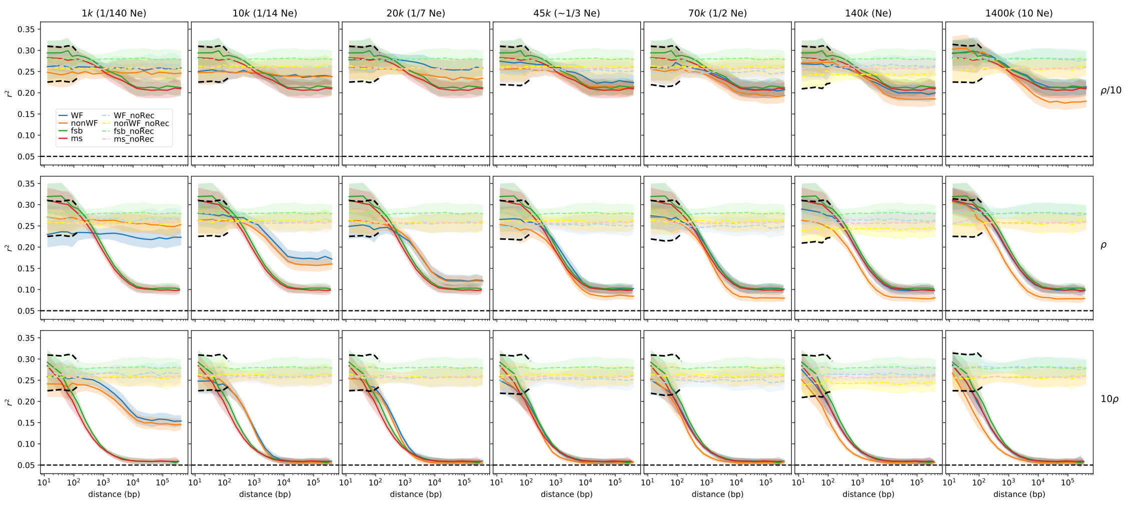

Figure 10: Linkage disequilibrium for WF and nonWF simulations with various number of forward generations and recombination rates (ρ). Ne=140 000, λ = 12200bp and RF=25. The horizontal dash line is the expected r^2 with free recombination when sampling 20 individuals (1/20). The shaded area represents the standard error of the mean, the standard deviation is 10 times larger (since we have 100 samples), see figure below.

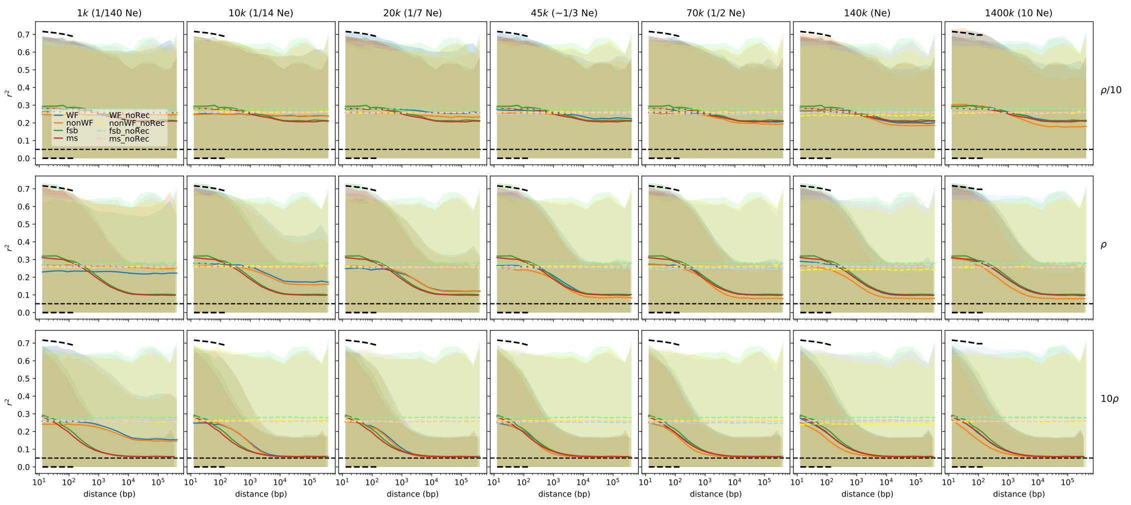

Figure 11: Same Figure as supplementary figure 10, but representing standard deviation instead of standard error of the mean.