1 **Title**: SimBit: A high performance, flexible and easy-to-use population

2 genetic simulator

3

4 **Author**: Remi Matthey-Doret[1,2]

5 1: Department of Zoology and Biodiversity Research Centre, University of British Columbia,

6 Vancouver, British Columbia V6T 1Z4, Canada

7 2: Institute of Ecology and Evolution, University of Bern, CH-3012 Bern, Switzerland

8

9 **Corresponding author**: remi.matthey-doret@iee.unibe.ch

1

10   **Abstract**

11   SimBit is a general purpose and high performance forward-in-time population genetics simulator.

12   SimBit has been designed to be able to model a wide diversity of complex scenarios from a simple

13   set of commands that are very flexible. SimBit also comes with a R wrapper that simplifies the

14   management of an entire research project from the creation of a grid of parameters and

15   corresponding inputs, running simulations and gathering outputs for analysis. Implementing

16   various representations of the individual's genotype allows SimBit to sustain a high performance

17   in a wide diversity of simulation scenarios. SimBit's performance was extensively benchmarked

18   in comparison to SLiM, Nemo and SFS_CODE. No single program systematically outperforms

19   the others but SimBit is most often the highest performing program and maintains high

20   performance in all scenarios considered.

Introduction

Evolutionary genetics has always had a strong theoretical background. As our understanding of ecological and evolutionary processes improves, we study more and more complex processes for which mathematical modelling becomes very tedious if not impossible. For such processes, only numerical simulations can allow us to perform realistic modelling. In fact, to my knowledge, the first work in computational biology has been conducted by one of the fathers of population genetics, Ronald A. Fisher (1950).

We are today in an uncanny valley in which we are almost able to perform realistic genome-wide simulations of populations but not quite yet. Individual-based simulations are used to investigate phenomena in evolutionary biology and ecology (e.g. Gilbert et al., 2017; Yeaman & Whitlock, 2011), to question conservation scenarios (e.g. Cowley, 2008; Halls & Welcomme, 2004) and are also used in statistical settings such as with Approximate Bayesian Computation (reviewed in Beaumont, 2010) or with a machine learning algorithm (e.g. Schrider & Kern, 2018). However, such technics are often computationally very expensive and it can take a lot of time to parametrize these simulations. As a consequence, many studies limit forward simulations to unrealistically low number of individuals or loci.

Writing an algorithm to make efficient individual-based simulations is no easy task, and most authors therefore rely on existing, flexible simulation programs. It is often difficult, however, to choose a simulation program. There are no objective ways to compare and express how user-friendly a program is. Also, different program packages have drastically different performance for different simulation scenarios. Learning how to use a new program can be a lengthy and difficult task, therefore many users just use the program they already know or just pick one program that is able to perform the simulations they need without questioning its performance. However, as shown

44   below, even under simple scenarios, a given program can be hundreds or thousands of times slower

45   than another one which will drastically affect the feasibility, or level of replication, of a study.

46   Here, I present SimBit, a general purpose forward-in-time population genetics simulator written

47   in C++. SimBit has been designed to have a high performance for a wide variety of simulation

48   scenarios. SimBit does so by using diverse representations of the genetic architecture for different

49   simulation scenarios. As a user of Nemo (Guillaume & Rougemont, 2006), SFS_CODE

50   (Hernandez, 2008; Hernandez & Uricchio, 2015) and SLiM (Haller et al., 2019; Haller & Messer,

51   2017, 2019), I gathered my experience to make SimBit a program that offers a fast learning curve

52   to new users. With a simple set of commands that are very flexible, users can quickly simulate a

53   great diversity of scenarios. SimBit can simulate a wide variety of selection scenarios (any

54   selection coefficient and dominance coefficient at any locus, any epistatic interaction with any

55   number of loci, any spatial and temporal changes of selection scenarios, etc.), demographic

56   scenarios (any number of discrete patches with specific migration scenario, hard vs. soft selection,

57   changes in patch size depending on fecundity, exponential vs logistic growth, gametic or zygotic

58   dispersion, etc.), mating systems (any cloning rate and selfing rate, hermaphrodites or males and

59   females), different types of representation of the genetic architecture (bi-allelic loci, QTLs, etc.)

60   and SimBit has a great diversity of tools to manipulate simulations and gather output. Finally,

61   SimBit comes with a R wrapper that is very handy for managing the creation of numerous input

62   commands. This article aims at presenting the general working of SimBit and compares its

63   performance to other similar programs. For detailed information about how to use SimBit, please

64   consult the manual.

65

66

67 **Demography and species ecologies**

68 In the current version, SimBit assumes non-overlapping generations (although different species

69 can have different generation times), diploidy (although one can mimic haploidy), and assumes

70 discrete patches (although patches can be made arbitrarily small, essentially mimicking continuous

71 space). Outside of these three assumptions, SimBit can simulate very diverse types of scenarios.

72 SimBit can simulate any number of patches with any migration matrix, carrying capacity, variation

73 of the patch size from the carrying capacity based on realized fecundity with exponential or logistic

74 growth model (the growth model can be set for each patch independently; see more on that below).

75 Each patch can be initialized at the desired size and all of the above parameters can vary over time.

76 Dispersal can happen at the gametic or at the zygotic phase and may be a function of the patch

77 mean fitness (hard versus soft selection). SimBit can also simulate multiple species and their

78 ecological interactions as explained below.

79

80 SimBit can simulate realistic changes in population in response to patch mean fitnesses. Let's

81 denote at time t the expected number of offspring of a species s produced in patch p as $\overline{P_{t,s,p}}$. Let's

82 also denote the patch growth rate $r_{t,s,p} = f \sum w_i$ as the product of f, the theoretical maximum

83 fecundity of an individual having a (relative) fitness of 1.0 (set by the user), and $\sum w_i$, the sum of

84 finesses in this patch. If the user allows the patch size to vary from the carrying capacity of this

85 species and that at time t, in patch p, for species s, the carrying capacity is set to $K_{t,s,p}$ then the

86 expected number of offspring produced is $\overline{P_{t,s,p}} = rN_{t,s,p}$ for the exponential model and $\overline{P_{t,s,p}} =$

87 $N_{t,s,p} + rN_{t,s,p}\left(1 - \frac{N_{t,s,p}}{K_{t,s,p}}\right)$ for the logistic model, where $N_{t,s,p}$ is the size of the patch p of species

88 s at time t. The actual number of offspring produced, $P_{t,s,p}$ can then either be set deterministically

89 $\left(P_{t,s,p} = \overline{P_{t,s,p}}\right)$ or stochastically $\left(P_{t,s,p} = Poisson(\overline{P_{t,s,p}})\right)$. With more than one patch, these

5

90   offspring produced are then spread out through migration. With a single patch (or in absence of

91   immigration and emigration for the patch p), $N_{t+1,s,p}$ is simply set to $P_{t,s,p}$.

92

93   Into the above framework, we can add the fact that different species can affect each other's through

94   their ecological relationships. This can be achieved through a "competition matrix" that

95   implements a Lotka-Volterra model of competition and/or through a "predation matrix" that

96   implements a consumer-resource model (or predator-prey model) with a linear rate of resource

97   consumption (introduction to these models in Otto & Day, 2007; discrete-time example of a

98   predator-prey model in Çelik & Duman, 2009). Let $\alpha_{i,s}$ be an element of the "competition matrix"

99   describing the competitive effect of species i on focal species s. The expected number of offspring

100   produced is then given by $\overline{P_{t,s,p}} = N_{t,s,p} + rN_{t,s,p}\left(1 - \frac{\sum_i \alpha_{i,s} N_{t,i,p}}{K_{t,s,p}}\right)$. Note that competitive effects

101   can only be set on species and on patches having logistic growth. Let $\beta_{i,s}$ be an element of the

102   "predation matrix" describing the effect of species i on species s. The predation effect is added to

103   the expected number of offspring produced $\overline{P'_{t,s,p}} = \overline{P_{t,s,p}} + \sum_i \beta_{i,s}$. In this last equation, I

104   assumed that all effects $\beta_{i,s}$ are independent of the patch sizes of both the causal and recipient

105   species but in practice a user can specify for each $\beta_{i,s}$ whether the effect should be multiplied by

106   the causal species patch size ($N_{t,i,p}$), by the recipient species patch size ($N_{t,s,p}$) or by both. SimBit

107   enforces that all the diagonal values $\alpha_{s,s} = 1.0$ and that all the diagonal values $\beta_{s,s} = 0.0$. SimBit

108   can also allow the patch size to overshoot the carrying capacity $K_{t,s,p}$ up to an arbitrary large value

109   allowing for oscillating or chaotic changes in patch sizes.

110

111

112

6

113 **Mating system**

114 SimBit can simulate hermaphrodites or males and females with an arbitrary sex-ratio. At every

115 reproduction event, an organism will be cloned with probability C and self with probability S. By

116 default, the cloning rate is set at 0.0 and the selfing rate is set at 1/2N (Wright-Fisher model), but

117 these can be set by the user.

118

119 **Types of loci and selection**

120 Different programs use different representations of the genetic variation. For example, Nemo

121 represents an individual's haplotype with an array in which the nth element of the array indicates

122 the allelic value for the nth locus. In SLiM, each individual's haplotype is represented with a

123 container of mutations (where each mutation is an object that stores its position and other

124 associated features as attributes). In SFS_CODE, a haplotype is represented with a linked list of

125 mutations. These different representations of the genetic variation have important consequences

126 for the performance of the software package. Nemo's technique is expected to perform well at high

127 genetic diversity per locus, while SLiM and SFS_CODE are expected to perform better at low

128 genetic diversity per locus. Nemo also has QTLs and SLiM can mimick QTLs through Eidos (the

129 programming language used to parameterize SLiM simulations). These different representations

130 also have consequences on the flexibility and performance of a program.

131

132 SimBit implements five different representations of the genetic variation called T1, T2, T3, T4 and

133 T5. I refer to these representations as types of loci. T1, T4 and T5 types of locus represent binary

134 loci. SimBit has multiple representations of binary loci in order to sustain flexibility and high

135 performance over a wide range of genetic diversity and of simulation scenarios. T2 type of locus

136   represents blocks that count mutations, T3 type of locus represent QTLs and all three types. More

137   information on these five types of representations is below. Loci of different types are integrated

138   on the same recombination map. The recombination rate can be specified between any pair of

139   adjacent loci (whether the two loci are of the same type or not) with any number of chromosomes.

140   Mutation rates can also be set independently for each locus.

141   For a number of types of loci (see below), SimBit can make use of an assumption about the

142   selection scenario that can provide substantial improvement in run time. I call this assumption the

143   "multiplicative fitness" assumption. The multiplicative fitness assumption assumes 1)

144   multiplicative fitness interactions among loci and 2) that the fitnesses of the three possible

145   genotypes at a given locus are 1, $1{-}s$ and $(1{-}s)^2$. When a user makes this assumption, SimBit

146   partitions a haplotype into blocks and computes the fitness value for each block. If, during

147   reproduction, no recombination events happen within a given block, then SimBit will not need to

148   recompute the fitness for this specific block as the fitness of the block can simply be multiplied by

149   the fitness of the same block on the other haplotype. By default, SimBit attempts to estimate the

150   optimal size of these blocks, but a user can also explicitly specify the position and location of each

151   block. This technique yields substantial performance improvement in terms of CPU time especially

152   when the recombination rate within blocks is relatively low (see 'Performance' section below).

153   Therefore, unless the exact dominance relationship is of central importance, it is generally

154   recommended to make use of this assumption.

155

156   The genetic architecture can be set independently for each species and all the selection scenarios

157   presented below can be set differentially for each species, habitat and time. By default, all of the

158   patches belong to the same habitat, but a user can assign each patch to a specific habitat and all the

8

159   selection pressures described below (including epistasis) can be specified for each habitat

160   independently. Also, selection can be applied on viability and/or on fertility.

161

162   *T1 loci*

163   T1 loci track binary variables (e.g., mutated vs wildtype). SimBit has in memory for each

164   haplotype an array of bits of the length of the number of T1 loci simulated. The nth bit indicates

165   whether the nth T1 locus of this haplotype is mutated or not. As such, T1 loci are somewhat similar

166   to Nemo's genetic representation. T1 loci have high performance for simulations with very high

167   per locus genetic diversity.

168   Selection scenarios on T1 loci are extremely flexible. A user can set the fitness values of each of

169   the three possible genotypes at each locus allowing for any kind of dominance scenario including

170   overdominance and underdominance. Any epistatic interactions between any number of loci can

171   also be specified. A user can also use the assumption of "multiplicative fitness" on T1 loci.

172

173   *T2 loci*

174   T2 loci are meant to represent aggregate blocks of loci, and, SimBit counts the number of mutations

175   happening in this block. This type should be used only when 1) the genetic diversity per T2 locus

176   is very high, 2) when performance is a major concern, 3) the user is satisfied with the limited

177   selection scenario it can model, and 4) a simple count of the number of mutations happening per

178   T2 locus for each haplotype is a sufficient output. Selection on T2 is forced to have multiplicative

179   effect among haplotypes (therefore T2 loci always use the assumption of "multiplicative fitness").

180

181

9

182   *T3 loci*

183   T3 loci are quantitative trait loci (QTL) and code for an *n*-dimensional phenotype. The user can

184   set the phenotypic effect of each T3 locus on each of the *n* axes of the phenotype, and these

185   phenotypic effects can also depend on the environment in order to simulate a plastic response. A

186   user can also add random developmental noise (drawn from a Gaussian distribution) in the

187   production of a phenotype in order to reduce heritability. For T3 loci, the user can define a fitness

188   landscape, where an individual's fitness is given by its phenotype.

189

190   *T4 loci*

191   For T4 loci, SimBit computes the coalescent tree of the population over time and adds the

192   mutations onto the tree when the user asks for output. As a consequence, T4 loci are necessarily

193   neutral. T4 loci are inspired from Kelleher et al. (2018) and the method has already been

194   implemented in SLiM (Haller et al., 2019). Tree recording technics can be very promising when

195   dealing with lots of highly linked neutral loci. This technic allows a forward-in-time simulator to

196   perform equally than backward-in-time simulators for some extreme simulation scenarios while

197   retaining many of the advantages of forward-in-time simulations such as simulating selection at

198   other loci (Haller et al., 2019).

199

200   *T5 loci*

201   T5 loci are very similar to T1 loci (two simulations with the same random seed differing only by

202   the fact that one uses T1 loci and the other uses T5 loci will produce the same output). For each

203   haplotype, SimBit has a dynamic sorted array with the position of each T5 locus that is mutated.

204   As such T5 loci are somewhat similar to how SLiM keeps track of its genetic architecture. With

205   high genetic diversity SimBit therefore tracks a lot of mutated loci, while with low genetic diversity

206   SimBit tracks few mutated loci. For this reason, T5 loci tend to perform better than T1 loci for

207   moderate to low genetic diversity per locus.

208   Behind the scene, SimBit will track separately T5 loci that are under selection and T5 loci that are

209   neutral for improved performance. SimBit can also compress T5 loci (either the neutral ones and/or

210   the selected ones) information in memory. Compression reduces the RAM usage by up to a factor

211   of 2 and can increase or decrease CPU time depending on the simulation scenario. By default,

212   SimBit makes this compression on the neutral T5 loci only and only when it is certain it will

213   improve performance. For advanced users, it is also possible to ask SimBit to invert the meaning

214   of some loci depending on their frequencies. For example, if the locus 23 is fixed or quasi-fixed,

215   haplotypes would track this 23rd locus only if they carry the non-mutated allele.

216   With T5 loci, one can specify the fitness values of the heterozygote and double mutants' genotypes

217   only allowing for all types of dominance including overdominance and underdominance. Just as

218   on T1 loci (and T2 loci), a user can take advantage of the assumption of "multiplicative fitness".

219

220   **Initialization**

221   Several options exist in SimBit to initialize and reset the genome of existing individuals. The patch

222   size as well as the genetic diversity for each locus can be set at initialization. A user can then

223   perform any mutation desired at predefined times with the option --resetGenetics. To ease user

224   interface, SimBit also allows the user to define "individual types" (via option --individualTypes).

225   Those individual types can then be used to either initialize a population or to insert (or replace)

226   new individuals into any patch at arbitrary moments (also via option --resetGenetics). One can, for

227   example, create individual types belonging to large hypothetical patches and simulate immigration

11

228   from these hypothetical patches by just introducing these individual types into the focal patch. This

229   speeds up simulations as SimBit does not explicitly simulate these large source patches.

230   It is also possible to start a simulation from the individuals of a previous simulation that have been

231   saved in binary files. Binary files are particularly useful to 1) avoid simulating a burn-in multiple

232   times, 2) resume a simulation from an intermediate timepoint, and 3) save the entire population in

233   a compact format to extract specific summary statistics later on.

234

235   **Outputs**

236   Outputs are often very limiting factors for population genetic simulators (Hoban et al., 2012).

237   SimBit can produce 30 different types of outputs (which can be sampled at any number of

238   generations throughout the simulation). These outputs include, but are not limited to, entire

239   genotypes of each individual in the metapopulation, allele frequencies, F_{ST}, VCF files, fitness

240   (specifying fitness for each type of locus), patch sizes, extinction times of the different species, the

241   whole genealogy between two specified generations, binary files of the entire population (that can

242   be reused for future simulations or simply to extract summary statistics later on). Many of these

243   outputs can be restricted on a specified subset of loci. SimBit can also simulate sequencing errors

244   before producing the outputs to make results easier to compare to empirical data.

245

246   **User interface**

247   SimBit reads options either directly from the command line or via an input file. An important goal

248   of SimBit is to have a user interface that takes input that is readable and in a very simple format to

249   give the users a good understanding of what they are simulating and offer very explicit error

250   messages when input is nonsense. SimBit recognizes specific options as they are proceeded by a

251   double dash (`--`). For example, `--patchCapacity unif 1e4` indicates that the carrying capacity is

252   uniform (keyword `unif`) for all patches and is set to 10,000. The ordering of these options does

253   not matter. SimBit also provides a number of macros that are mainly inspired from R functions.

254   These inputs can be read either directly from the command line or from a file. SimBit also comes

255   with an R wrapper.

256   In order to be fast and easy to learn, SimBit provides many functionalities with a relatively small

257   number of options. It achieves this by having most options being specific to a generation, a habitat

258   and/or a species and uses specific markers, @G, @H and @S to input information that are

259   generation-specific, habitat-specific and species-species, respectively. For example, the entry `--`

260   `N @G0 unif 100 @G5e3 unif 1000` asks for the carrying capacity of all patches to be

261   uniformly (keyword `unif`) set to 100 from generation 0 to generation 4999 and then set to 1000

262   until the end of the simulation. Also, most options come with a diversity of modes of data entry.

263   For example, for the migration scenario, a user can indicate the whole dispersal matrix or can

264   simply specify an island model, a linear stepping stone model or a Gaussian dispersal kernel.

265   Below, I benchmark SimBit in comparison to other softwares. Examples of command line inputs

266   to SimBit for these simulations which results are shown on figures 1, 2, S1, S2, S3 and S4 as well

267   as for the simulations of figure 3 are found in appendix A. Here is an example of a input file used

268   for this benchmark. Please see manual for more information.

13

269

```
##############################
### Example of input file ###
##############################

### Number of patches
--PatchNumber 1

### Carrying capacity
--N unif 1e5

### Genetic architecture. Asks for 60000 T5 loci
--Loci T5 6e4

### Mutation rate on T5 loci
--T5_mu unif 1e-7

### Selection (uses multfit assumption)
--T5_fit multfitUnif 0.99999

### Recombination rate
# Values are interpreted as a "rate".
# For centimorgan, use "cM", instead of "rate"
--r rate unif 1e-7

### Number of generations
--nbGens 1e5
```

270

271

272   SimBit also comes with an R wrapper that is particularly useful for building numerous input

273   simulations. Without going into explaining the detail working of the wrapper, let's consider a

274   complete example of code that will test how different migration rates and number of patches in an

275   island model affect F_{ST}. The first step is to create a grid of parameters (a "data.frame"), where each

276   row contains information for a single simulation. We will use a full factorial design with three

277   distinct migration rates and seven distinct number of patches. We will run 20 replicates for each

278   of these 3×7=21 combinations resulting in a grid of parameters of 420 rows. The argument

14

279   "outputFilePrefix" sets a column called "outputFile" with the prefix given followed by the row

280   number. This column will be used to set the where outputs should be directed.

```
#########################
## Load SimBitWrapper ##
#########################

# devtools::install_github("RemiMattheyDoret/SimBitWrapper")
require(SimBitWrapper)

##############################
## Create grid of parameters ##
##############################

parameterGrid = fullFactorial(
    PatchNumber = c(2,3,4,5,6,7,8),
    migrationRate = c(0.001, 0.003, 0.01),
    N = 1e3,
    nbLoci = 1e4,
    nbGenerations = 5e4,
    recRate = 1e-4,
    mu = 1e-5,
    replicate = 1:20,
    outputFilePrefix = "/Users/Remi/mySims/output_"
)
```

281

282

283   The second step is to loop through the rows of the parameter grid in order to run the simulations

284   (or to create the input file to run them later on). For this, we use the function

285   GetParameterGridData, which, for each column of the grid of parameters, sets a variable

286   with name equal to the column name and value equal to the value of this column at the specified

287   row of the specified parameter grid given in input.

```
########################################
## Create inputs and run simulations ##
########################################

for (row in 1:nrow(parameterGrid))
{
    ### Get data for the row
    GetParameterGridData(parameterGrid, row)

    ### Initialize the input
    input = Input$new()

    ### Set the values
    input$set("PatchNumber", PatchNumber)
    input$set("m","island", migrationRate)
    input$set("N", "unif", N)
    input$set("nbGenerations", nbGenerations)
    input$set("L", "T1", nbLoci)
    input$set("T1_mu", "unif", mu)
    input$set("r", "rate", "unif", recRate)
    input$set("T1_FST_file", outputFile, nbGenerations)

    ### Run the simulation
    input$run(maxNbThreads=24)
}
```

288

289

290   The argument `maxNbThreads` is an easy way to parallelize the simulations.

291   `maxNbThreads=24` does not mean that a given simulation will use 24 threads (each simulation

292   takes one thread) but that the `run` method will start 24 simulations in the background and will

293   then wait that one of them finishes before starting a 25th simulation. Please see manual for further

294   information about the `run` method. It is sometimes more practical to print the input command into

295   a file either and run the simulations from the shell at a later time. This can be achieved with

296   `input$print("/path/to/input.txt")`. Finally, the last step is to gather the outputs and

297   graph the results. In order to gather the outputs, we use the function `gatherData`. This function

16

298   uses a number of optional parameters (see manual) but default parameters work fine for our simple

299   example.

```
###############################
## Gather and graph outputs ##
###############################

### Gather simulation outputs
data = gatherData(parameterGrid)

### Graph
ggplot2::ggplot(data, aes(y=FST_WeirCockerham_ratioOfAverages,
x=PatchNumber, color=as.factor(migrationRate))) +
stat_summary() + theme_classic()
```

300

301   In this simple example, the entire study (defining the parameters, creating the inputs, running the

302   simulations, gathering and graphing the results) takes 16 lines of code (16 expressions; including

303   loading packages, excluding the curly braces; and it could be reduced to 7 lines only)! The column

304   "FST_WeirCockerham_ratioOfAverages" used for plotting corresponds to Weir & Cockerham

305   (1984) estimator of F_{ST}. The resulting graph is displayed in figure S5 on which is added the

306   theoretical expected F_{ST} values from Charlesworth (1998) for comparison.

307
308   **Program comparison – Performance**

309   It is often hard for a user to know which program to use for a given study. Indeed, few articles

310   compare program's features (but see Hoban, 2014, who compares software flexibility), and when

311   authors publish a new program, they do not always compare its performance to other similar

312   programs (but see performance comparisons between SLiM, SFS_CODE and fwdpp in Haller &

313   Messer, 2017).

314

17

315   In this article, I compared performance of SimBit to three forward-in-time programs; SFS_CODE

316   (Hernandez, 2008; Hernandez & Uricchio, 2015), SLiM (Haller et al., 2019; Haller & Messer,

317   2017, 2019; Messer, 2013) and Nemo (Guillaume & Rougemont, 2006). I chose these three

318   programs because they are all forward-in-time simulation platforms, they can all simulate

319   selection, they are all popular (392 citations among the articles announcing SLiM, SLiM2, SLiM3

320   and the implementation of tree recording sequences in SLiM; 127 citations for Nemo; 216 citations

321   for SFS_CODE; as of 23rd April 2020 on Google Scholar) and are generally considered the highest

322   performing software available.

323

324   SimBit contains a number of options that are meant to refine its performance (see section

325   "Performance options" in the manual). In practice though, most users will probably only need to

326   choose the type of loci to simulate, and SimBit will do a decent job to figure out how best to

327   simulate it. In order to best represent the performance that a new user ought to expect from SimBit,

328   however, all simulation performances (CPU time and memory usage) presented below are made

329   with the default parameters of SimBit.

330

331   In order to compare program performance, I ran basic simulations with a single Wright-Fisher

332   population, uniform mutation rate and a uniform recombination rate. All loci experienced a

333   selection coefficient of s=0.00001 and h=0.5. Low selection coefficients were chosen to 1) prevent

334   any software from throwing an error stating that it might suffer from round-off errors caused by

335   low mean fitness and 2) reduce the effects of assuming multiplicative fitness among haplotypes on

336   the simulated scenario (fitness differences between simulations that take advantage of the

337   assumption of multiplicative fitness and the ones that do not is of the order of 10^{-11}). Note that

18

338   while SimBit can take advantage of this assumption of multiplicative fitness on demand,

339   SFS_CODE is forced to make this assumption and Nemo and SLiM cannot take advantage of this

340   assumption. I varied the mutation rate (taking values 10^{-7}, 10^{-5} and 10^{-3} per locus), the

341   recombination rate (taking values 0, 10^{-9} and 10^{-7} and 10^{-5} per adjacent locus), the carrying capacity

342   (taking values 10^2, 10^3, 10^4, 10^5 and 10^6 diploid individuals), and the number of loci (taking values

343   6, 6×10^2, 6×10^4 and 6×10^6) in a full factorial design. All simulations ran for 10,000 generations. I

344   ran these simulations with Nemo (version 2.3.46), SLiM (version 3.1), SFS_CODE (version

345   20150910) and SimBit (version 4.11.0). Because using Nemo's full potential is not trivial, for

346   Nemo, the input files used for these benchmarks were directly created by Frederic Guillaume. In

347   order to compare the behaviour of different types of loci and selection scenarios in SimBit, I ran

348   all simulations four times in SimBit with T1 and T5 types of loci with and without making use of

349   the assumption of multiplicative fitness among haplotypes. CPU time and peak in Resident Set

350   Size (RSS; memory) usage are reported. Simulations that exceeded 10 days (240 hours) of

351   simulation time or 20GB of memory usage were killed and are reported below with a dot at 240

352   hours (8.64×10^5 seconds in the units used on the figures) and at 20GB (2×10^7 kb in the units

353   used on the figures). All these simulations were run on an Intel Xeon X5650 processor and codes

354   were compiled with gcc-4.8.2rev203690. I ensured that the number of SNPs were not significantly

355   different between all four programs for three of the simulation scenarios benchmarked.

356   For brevity and because changing the recombination rate has very little effect on the results (only

357   SFS_CODE appears to significantly slow down with higher recombination rates), I am showing

358   only the recombination rate 10^{-7} and only the carrying capacities 10^3, 10^4 and, 10^5 in the main

359   figures. The other benchmarks are found in supplementary material. Figure 1 compares the CPU

360   time among SimBit simulations (T1 vs. T5 and with vs. without taking advantage of the

19

361   assumption of multiplicative fitness among haplotypes) for a subset of scenarios. Figure S1 and

362   S2 compare, respectively, the CPU time and the memory usage among SimBit simulations for all

363   scenarios. Figure 2 compares CPU time among Nemo, SLiM, SFS_CODE and SimBit for a subset

364   of scenarios. Figure S3 and S4 compare, respectively, the CPU time and the memory usage among

365   Nemo, SLiM, SFS_CODE and SimBit.

366

367   **Figure 1:** Comparison of computational time among the four different ways to simulate the same

368   evolutionary scenario using SimBit. Results here are only for a subset of parameters (excluding

369   N=100, N=10^6 and all scenarios where the recombination rate among adjacent loci differs from

370   10^{-7}). Other scenarios are in figure S1. Comparisons of memory usage (max Resident Set Size) are

371   found in figure S2. Simulations that exceeded 10 days (240 hours) of simulation time or 20GB of

372   memory were killed and are reported below with an empty dot at 240 hours (8.64×10^5 second).

373   The bold **M** signifies the usage of the assumption of multiplicative fitness.


374

375

376

21
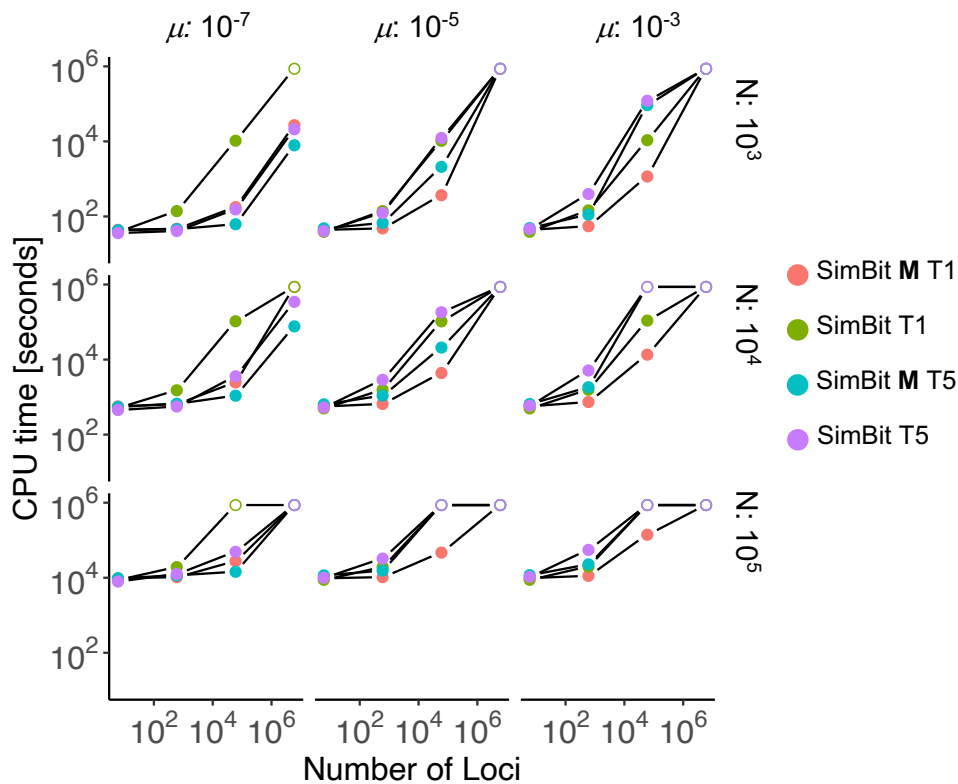
377   As expected, T1 loci perform best at high per locus genetic diversity, while T5 loci perform best

378   at moderate to low per locus genetic diversity (figure 1). This is because with T5 loci, SimBit

379   tracks the mutated loci, while with T1 loci, SimBit tracks every locus whether mutated or not (see

380   above section "Representations of the genetic architecture").

381   Simulations taking advantage of the assumption of multiplicative fitness generally performed

382   better. This advantage decreases as recombination gets higher. For the range of recombination

383   rates explored (up to 10^{-5} among adjacent loci), simulations taking advantage of the assumption of

384   multiplicative fitness always outperformed the simulations that did not make this assumption. The

385   reason why recombination rate matters for performance is because, as explained in section "Types

386   of loci and selection", SimBit needs to recompute fitness for a fitness block only if a recombination

387   event happens within this block when using the multiplicative fitness assumption.

388

389

22

390 **Figure 2:** Comparison of computational time among the four different simulation programs Nemo,

391 SFS_CODE, SLiM and SimBit. For SimBit, two lines are displayed showing the best performing

392 between T1 and T5 loci from figure 1, once taking advantage of the assumption of multiplicative

393 fitness, once without taking advantage of this assumption. For comparison, SLiM and Nemo are

394 unable to take advantage of this assumption while SFS_CODE is forced to make this assumption.

395 Other scenarios are in figure S3. Comparisons of memory usage (max Resident Set Size) are found

396 in figure S4. See figure 1 for more details.

397
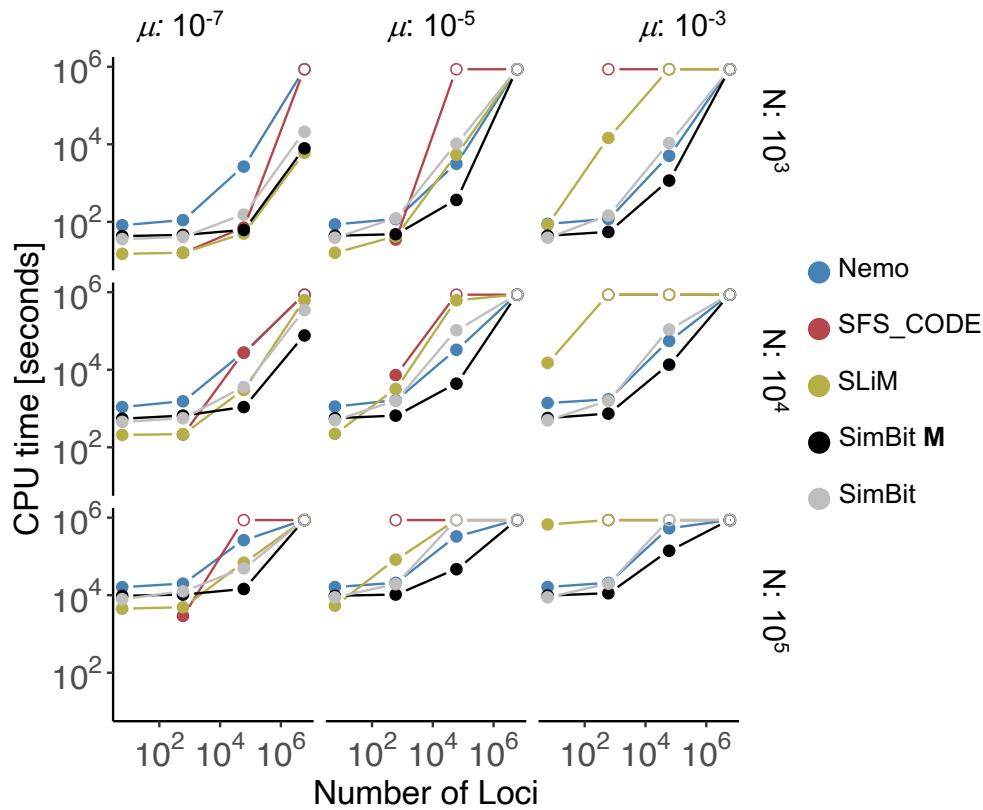
398

399

400   Comparisons between different programs highlight that there is no one program that always

401   performs best (figure 2; figure S3). However, unlike all other software tested, SimBit perform

402   highly in all simulation scenarios considered. SFS_CODE's CPU time and peak RSS increases

403   exponentially with increase in mutation rate and population size (see also simulations performed

404   by the Ryan Hernandez on SFS_CODE websites;

405   sfscode.sourceforge.net/SFS_CODE/Performance.htlm). Hence, SFS_CODE performs well for

406   simulations that have very low genetic diversity, but it quickly becomes very slow as genetic

407   diversity increases.

408   Nemo is most competitive when there is high genetic diversity per locus (high mutation rate and

409   high population size). This was expected because Nemo tracks every single locus for each

410   haplotype whether or not it is mutated. In fact, with high genetic diversity, Nemo sometimes runs

411   in less time than SimBit when SimBit did not take advantage of the multiplicative fitness

412   assumptions (the grey dots in figures 2 and S3). Nemo never outperformed SimBit in terms of

413   memory usage though (Figure S4) or in terms of CPU time when SimBit takes advantage of the

414   multiplicative assumption.

415   SLiM, just like SFS_CODE, performs best at very low genetic diversity. SLiM computational time

416   is however not as exponential as SFS_CODE, which makes SLiM fast for a wider range of

417   simulation scenarios. SLiM tends to perform better than SimBit when there is little genetic

418   diversity, while SimBit tends to perform better when there is moderate to high genetic diversity.

419   In general, performance comparison in terms of memory usage (figures S2, S4) mirrors well the

420   performance comparisons in terms of CPU time (figures S1, S3).

421   A difference in performance is not just a question of whether a user will have to wait a little longer

422   to get their output; often it is the difference between a research project that is feasible or not. The

24

423   log scale on figures S1 and S2 (and supp. figures) might give the reader a false impression of the

424   importance of an observed difference. Consider for example the simulation scenario where $r=10^{-7}$,

425    $N=10^3$, $\mu=10^{-7}$ and 6 loci where SLiM outperforms SimBit. SLiM runs in 16 seconds while

426   SimBit runs in 37 seconds. Let's now consider the simulation scenario where $r=10^{-7}$, $N=10^5$, $\mu=10^{-7}$

427   and 6×10^4 loci. SimBit (with multiplicative fitness assumption) runs in ~4 hours, while SLiM

428   runs in ~19 hours, Nemo runs in more than 3 days and SFS_CODE does not manage to finish

429   within the 10-day limit. To further consider comparisons between SLiM and SimBit as example,

430   from figure 2, the simulation scenario where SLiM is comparably the fastest, SLiM is 2.56 times

431   faster than SimBit; SimBit took 41 seconds while SLiM took only 16 seconds. For the simulation

432   scenario where SimBit is comparably the fastest, SimBit is (at least) 1169 times faster than SLiM;

433   SimBit took ~12.3 minutes while SLiM was killed after overpassing the 240 hours walltime. These

434   performance differences can translate into a major determinant of what can be achieved for a

435   research project.

436

437   These very simple simulation scenarios benchmarked above might not be representative of what

438   people really want to simulate. I therefore performed further benchmarking by comparing the

439   performance of Nemo, SLiM, SFS_CODE and SimBit for simulations inspired by recent papers.

440   I sampled three papers, one that performed simulations with SFS_CODE (O'Neill et al., 2019),

441   one that performed simulations with Nemo (Gilbert et al. 2017) and one that performed simulations

442   with SLiM (Booker & Keightley, 2018). To simplify the writing of the commands and make sure

443   that the comparison is fair, I simplified the Booker and Keightley (2017) simulations by assuming

444   a constant mutation rate and recombination rate and used the gamma distribution of fitness effects

445   with a mean of 0.05 and an alpha parameter of 0.111. For the Gilbert et al. (2017) paper, the

446   simulations have also been slightly modified from the original. The original paper's specified a

447   "breeding kernel" that can only run on a modified version of Nemo that is not directly published

448   on Nemo's official website. Hence, for the Gilbert et al. (2017) simulation, I removed the

449   breeding_kernel and modified the size of the dispersal kernel appropriately. For simplicity

450   (because the original input file was 390Mb large), I also used a linear stepping stone model of

451   8000 patches starting with the 1000 left-most patches at carrying capacity and the others empty. I

452   made sure the expansion speed was similar among the two programs. For fairness, I compared the

453   Nemo and SLiM that cannot take advantage of the assumption of multiplicative fitness with SimBit

454   that does not make this assumption, while I compared SFS_CODE that is forced to make this

455   assumption with SimBit that makes this assumption. I also performed a benchmark inspired from

456   human genome and human ancestral demography. I simulate 500 patches of 100 individuals each

457   in a linear stepping stone model with a migration rate to either of the two neighboring patch of 0.2.

458   The genome contained 2×10^8 sites with a uniform mutation rate of 2×10^{-8} and a uniform

459   recombination rate of 10^{-8}. For simplicity, all loci were under purifying selection with a constant

460   selection coefficient of 0.0001 and a dominance coefficient of 0.5. Finally, I added a benchmark

461   of a simple Wright-Fisher simulation scenario (N=1000, $\mu=10^{-5}$, 10^6 loci, r=0; 5000 generations)

462   without selection. Neutral loci can be tracked through a coalescent tree for both SLiM (with Tree

463   Recording and subsequent analysis of the outputted binary file in Python) and SimBit (with T4

464   loci). These simulations were run on an Intel i7-8559u processor, and codes were compiled with

465   clang-800.0.42.1.

466   SimBit systematically outperforms the software used in the original papers (figure 3). For the

467   simulation inspired from human genetics and ancestral human population, SimBit outperformed

468   SLiM whether it made use of the multiplicative fitness assumption or not. Finally, for the "Neutral

469   simulation example", the coalescent tree recording technique of both SLiM and SimBit vastly

470   outperform more traditional techniques (figure 3). With "traditional techniques", SLiM, Nemo and

471   SimBit took 8m29s, 4m05s and 1m18s, respectively, while using coalescent tree recording

472   methods, SLiM and SimBit only took 16.6 seconds and 1.2 seconds, respectively. Here, I only

473   considered an extreme scenario to exemplify the possible advantage of tree recording techniques.

474   For example, I used a recombination rate of zero. With higher recombination rates, the

475   computational time of tree recording techniques would become slower, while it would not have

476   much impact on the runs that did not use a tree recording technique.

477

27

478   **Figure 3:** Comparison of CPU time among the four programs to reproduce simulations inspired

479   from three recent papers as well as for a neutral simulation scenario with extreme parameters

480   chosen to highlight the possible advantage of T4 loci (Tree recording). The bold **M** signifies the

481   use of the assumption of multiplicative fitness. SFS_CODE simulation from the "Neutral

482   simulation example" as well as both SFS_CODE and Nemo simulations from the "Human

483   ancestral populations" were purposely killed after overpassing 50 times SimBit's CPU time for

484   the same simulation.
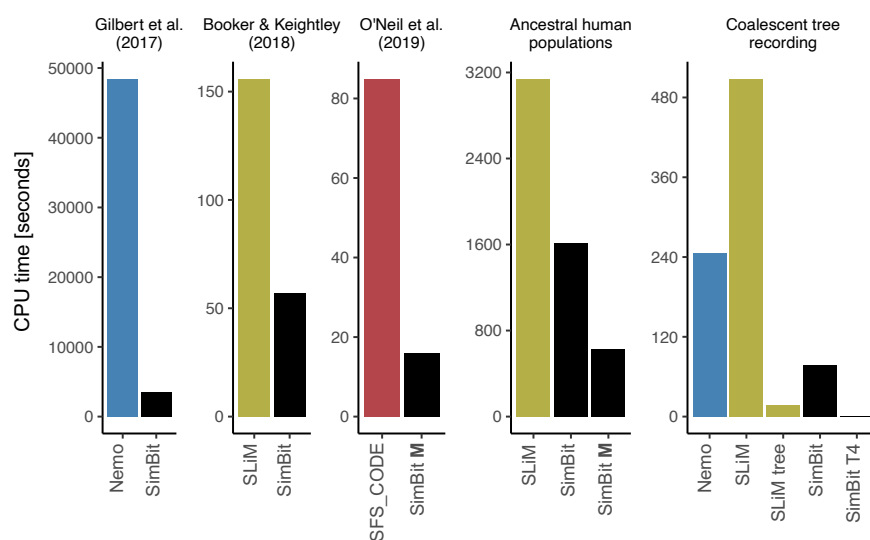
485

486   **Conclusion**

487   There is no perfect way to compare program performance, and one must always be careful when

488   making conclusions from such a benchmark. First, the parameter space considered is, of course,

489   finite. For example, my benchmark does not include any single-locus simulations, simulations with

490   high selfing rates or with males and females instead of hermaphrodites, or any simulations with a

491   very high recombination rate. Also, different programs mean different things by a locus.

492   SFS_CODE simulate triplets of loci as a codon. This means that many mutations that are

493   happening in SFS_CODE are synonymous mutations that don't affect fitness. Consequently, the

28

494   performance comparisons shown here are unfairly favourable to SFS_CODE compared to Nemo,

495   SLiM and SimBit, but it would not be any fairer either to run all SFS_CODE simulations with

496   three times as many loci. Nemo uses a byte to represent each neutral locus (but only a single bit

497   for loci under selection) hence allowing for the representation of up to 256 possible alleles at

498   neutral loci. SimBit on the other hand represent each locus with a single bit (whether the locus is

499   under selection or not), hence allowing for only two possible alleles. SLiM's mutations "stack"

500   (no reverse mutations) at a given locus, hence simulating a pseudo infinite allele type of model

501   (see   SLiM   manual   on   "mutation   stacking"   for   more   information;

502   http://benhaller.com/slim/SLiM_Manual.pdf). As explained above, SimBit contains a number of

503   performance tweaks a user can take advantage of to improve the performance above the default

504   run mode (compression of T5 data in memory, allowing inversion of the meaning of T5 loci

505   depending on their frequency, turning on/off the swapping of pointers for haplotypes that do not

506   recombine or mutate during reproduction, setting manually the positions of blocks for the

507   multiplicative fitness assumption). However, the above simulations were all performed with

508   SimBit default values for these performance tweaks, which is somewhat unfair to SimBit.

509

510   SimBit has already been used in a number of projects. It has been used for simulations that require

511   very high performance, simulating the effect of background selection of large stretch of DNA in

512   structured populations (Matthey-Doret & Whitlock, 2019). SimBit has also been used for two

513   projects on genetic rescue, one requiring habitat-specific epistatic interactions (Nietlisbach et al.,

514   forthcoming) and one requiring complex metapopulation initialization and introduction of

515   predefined individuals during the simulation (Whitlock lab consortium, forthcoming). SimBit is

516 under a permissive free program license and is available at

517 https://github.com/RemiMattheyDoret/SimBit.

518

Acknowledgment

519

520 Thank you to Michael C. Whitlock for his help through discussions in designing SimBit. Thank

521 you to both Michael C. Whitlock and Kimberly J. Gilbert for helpful comments on the manuscript.

522 Thank you to Pirmin Nietlisbach for being the main beta tester and for his advice on how to

523 improve the user interface and the manual. Thank you also to Ben Haller and Frédéric Guillaume

524 for their feedback on how to make a fair comparison among programs. Special thanks to Frédéric

525 Guillaume for his help at creating the input files for Nemo and for his feedback about how to

526 display the benchmark results in a way that is fair. Finally, thank you to ComputeCanada for the

527 computational resources used for benchmarking.

528

Funding

529

533

534   Beaumont, M. A. (2010). Approximate Bayesian Computation in Evolution and Ecology. *Annual*

535   *Review of Ecology, Evolution, and Systematics*, *41*(1), 379–406.

536   https://doi.org/10.1146/annurev-ecolsys-102209-144621

537   Booker, T. R., & Keightley, P. D. (2018). Understanding the Factors That Shape Patterns of

538   Nucleotide Diversity in the House Mouse Genome. *Molecular Ecology*, 18.

539   https://doi.org/10.1093

540   Çelik, C., & Duman, O. (2009). Allee effect in a discrete-time predator–prey system. *Chaos,*

541   *Solitons & Fractals*, *40*(4), 1956–1962. https://doi.org/10.1016/j.chaos.2007.09.077

542   Charlesworth, B. (1998). Measures of divergence between populations and the effect of forces

543   that reduce variability. *Molecular Biology and Evolution*, *15*(5), 538–543.

544   https://doi.org/10.1093/oxfordjournals.molbev.a025953

545   Cowley, D. E. (2008). Estimating required habitat size for fish conservation in streams. *Aquatic*

546   *Conservation: Marine and Freshwater Ecosystems*, *18*(4), 418–431.

547   https://doi.org/10.1002/aqc.845

548   Gilbert, K. J., Sharp, N. P., Angert, A. L., Conte, G. L., Draghi, J. A., Guillaume, F., Hargreaves,

549   A. L., Matthey-Doret, R., & Whitlock, M. C. (2017). Local Adaptation Interacts with

550   Expansion Load during Range Expansion: Maladaptation Reduces Expansion Load. *The*

551   *American Naturalist*, *189*(4), 368–380. https://doi.org/10.1086/690673

552   Guillaume, F., & Rougemont, J. (2006). Nemo: An evolutionary and population genetics

553   programming framework. *Bioinformatics*, *22*(20), 2556–2557.

554   https://doi.org/10.1093/bioinformatics/btl415

555   Haller, B. C., Galloway, J., Kelleher, J., Messer, P. W., & Ralph, P. L. (2019). Tree-sequence

556   recording in SLiM opens new horizons for forward-time simulation of whole genomes.

557   *Molecular Ecology Resources*, *19*(2), 552–566. https://doi.org/10.1111/1755-0998.12968

558   Haller, B. C., & Messer, P. W. (2017). SLiM 2: Flexible, Interactive Forward Genetic

559   Simulations. *Molecular Biology and Evolution*, *34*(1), 230–240.

560   https://doi.org/10.1093/molbev/msw211

561   Haller, B. C., & Messer, P. W. (2019). SLiM 3: Forward Genetic Simulations Beyond the

562   Wright–Fisher Model. *Molecular Biology and Evolution*, *36*(3), 632–637.

563   https://doi.org/10.1093/molbev/msy228

564   Halls, A. S., & Welcomme, R. L. (2004). Dynamics of river fish populations in response to

565   hydrological conditions: A simulation study. *River Research and Applications*, *20*(8),

566   985–1000. https://doi.org/10.1002/rra.804

567   Hernandez, R. D. (2008). A flexible forward simulator for populations subject to selection and

568   demography. *Bioinformatics*, *24*(23), 2786–2787.

569   https://doi.org/10.1093/bioinformatics/btn522

570   Hernandez, Ryan D., & Uricchio, L. H. (2015). *SFS_CODE: More Efficient and Flexible*

571   *Forward Simulations* [Preprint]. Bioinformatics. https://doi.org/10.1101/025064

572   Hoban, S. (2014). An overview of the utility of population simulation software in molecular

573   ecology. *Molecular Ecology*, *23*(10), 2383–2401. https://doi.org/10.1111/mec.12741

574   Hoban, S., Bertorelle, G., & Gaggiotti, O. E. (2012). Computer simulations: Tools for population

575   and evolutionary genetics. *Nature Reviews Genetics*, *13*(2), 110–122.

576   https://doi.org/10.1038/nrg3130

577   Kelleher, J., Thornton, K. R., Ashander, J., & Ralph, P. L. (2018). Efficient pedigree recording

578   for fast population genetics simulation. *PLOS Computational Biology*, *14*(11), e1006581.

579   https://doi.org/10.1371/journal.pcbi.1006581

580   Matthey-Doret, R., & Whitlock, M. C. (2019). Background selection and F_{ST}: Consequences for

581   detecting local adaptation. *Molecular Ecology*, *28*(17), 3902–3914.

582   https://doi.org/10.1111/mec.15197

583   Messer, P. W. (2013). SLiM: Simulating Evolution with Selection and Linkage. *Genetics*,

584   *194*(4), 1037–1039. https://doi.org/10.1534/genetics.113.152181

585   O'Neill, M. B., Shockey, A., Zarley, A., Aylward, W., Eldholm, V., Kitchen, A., & Pepperell, C.

586   S. (2019). Lineage specific histories of *Mycobacterium tuberculosis* dispersal in Africa

587   and Eurasia. *Molecular Ecology*, mec.15120. https://doi.org/10.1111/mec.15120

588   Otto, S. P., & Day, T. (2007). *A biologist's guide to Mathematical Modeling in Ecology and*

589   *Evolution*. Princeton University Press.

590   Schrider, D. R., & Kern, A. D. (2018). Supervised Machine Learning for Population Genetics: A

591   New Paradigm. *Trends in Genetics*, *34*(4), 301–312.

592   https://doi.org/10.1016/j.tig.2017.12.005

593   Weir, B. S., & Cockerham, C. C. (1984). Estimating F-Statistics for the Analysis of Population

594   Structure. *Evolution*, *38*(6), 1358–1370.

595   Yeaman, S., & Whitlock, M. C. (2011). THE GENETIC ARCHITECTURE OF ADAPTATION

596   UNDER MIGRATION-SELECTION BALANCE: THE GENETIC ARCHITECTURE

597   OF LOCAL ADAPTATION. *Evolution*, *65*(7), 1897–1911.

598   https://doi.org/10.1111/j.1558-5646.2011.01269.x

599