SimBit

Simulating Individual Markers with Bits

User Manual

Author:
Remi Matthey-Doret
Important contributor:
Michael Whitlock
Main beta tester:
Pirmin Nietlisbach

Last revised December 17, 2018, for version 4.0.6

Contents

1 Correspondance

I would be happy to hear about your questions, bug reports or requests for new features. If you receive an error message starting by "Internal error", then please send me an email with the error message, SimBit's version and the input data. Please, when applicable, always make sure to provide a reproducible example of your problem (input data, SimBit version) and report the error message.

Remi Matthey-Doret
remi.b.md@gmail.com
matthey@zoology.ubc.ca
Beaty Biodiversity Research Center, room 205
Dept. of Zoology, University of British Columbia
6270 University Blvd, Vancouver, BC, V6T 1Z4
Phone: +1 (604) 369-5929

2 Getting started

2.1 SimBit in a few words

SimBit is a flexible and fast forward in time simulation platform for finite site mutation models with arbitrary genetic architecture, selection scenario (incl. local selection, epistasis and all possible types of dominance) and demography. SimBit has been created with the purpose of performing realistic simulations of large portion of chromosomes in little time with little RAM usage. SimBit's flexibility and performance has been shown to be a useful tool for a diversity of interest. Depending on the selection scenario that is being inputted, SimBit will select the fastest way to simulate it.
SimBit has first and foremost been created to be extremeley fast and RAM efficient when dealing with large genomes. When using a selection scenario, please consider that accepting the assumption that allelic effect are multiplicative (the fitness of the three possible genotypes of a diploid individual is 1, 1-s and $(1-s)^2$) will drastically reduce the computational time. Today, SimBit became a flexbile tools allowing for example to simulate several species and their ecological interactions.

2.2 How to obtain SimBit

SimBit is available on GitHub at *https://github.com/RemiMattheyDoret/SimBit*. If you are experiencing trouble downloading from GitHub, you might want to have a look at the following link .

2.3 How to compile SimBit

Using the command line (terminal), `cd` to the downloaded directory, notice the presence of `Makefile` and do `make`. This is it! You should now have an executable called `SimBit` in `/bin`.
By default, the Makefile is using the standard C++ of 2014 (c++14) or any more recent standards.

3 A few a priori information

3.1 How to give arguments to SimBit

SimBit works through the command line. To use SimBit, just call the executable and follow it with options starting with the prefix `--` (double dash). Each option is followed by an entry. For example `--nbGenerations 1000` indicates that you want a simulation lasting for 1000 generations. SimBit will list all available options if you just call the executable without giving it any arguments.

```
./SimBit --option1 arguments for option 1 --option2 arguments for option
 2
```

For some options, there exists a short and a long name (or even several long names). For example to set the carrying capacity per patch the short name is `--N` and the long name is `--PatchCapacity`. In this manual, I will indicate such options as `--shortName (--LongName1, --LongName2)` as for example `--N (--PatchCapacity)`. In general, I will prefer the usage of short names over long names.
SimBit will reorder the options you input. As such, you do not need to bother about the ordering. If an option is missing SimBit will use the default when a default is available. If no default is available for a missing option, if an option is present more than once, if two entries do not coincide or if an entry is none-sense, then SimBit will throw an (hopefully explicit enough) error.
It is also possible to put all arguments (same format) in a file and specify the file

and the line of the file that contains the argument. If the argument file is called `ArgFile.txt` and the arguments of interest are found in the 12th line (the first line is numbered 1, not 0), then one can do
./SimBit file ArgFile.txt 12
instead of `file`, one can equivalently just write `F`, `f` or `FILE`. Note that, to make the input commands easier to look at, one can escape newline characters in the file.

3.2 Presentation of options in the manual

When options have a short and a long name I will refer to them as `--ShortName (--LongName)`. Some options expect you to input a `Unit`, a `Mode` (Mode of entry or ways you input the information as the user). When the option expects one or more integer values (e.g. `--PatchNumber`), I use `int`. When the option expects one or more float values (e.g. `--T1_mu`) I use `float`. When the type of entries (`int` vs `float`) varies depending on the Mode, I use `value`.

3.3 The three basic types of loci

SimBit contain three types of loci, type 1 called "T1", type 2 called "T2" and type 3 called "T3".

3.3.1 T1 loci

Each T1 locus is a coded as a single bit and the site can therefore take only two states 0 and 1 (or wildtype allele and mutated allele if you prefer). A T1 locus can be thought as a single nucleotide site (although it can take only two states and not four) or as any arbitrary longer stretch of DNA. The selection pattern that can apply to T1 loci are numerous including local selection, all types of dominance (incl. overdominance and heterozygote disadvantage) and any epistatic relationships among any number of T1 loci.
A specific option exists for the initialization of T1 loci (see below `--T1_ini (--T1_Initial_AlleleFreqs)`).

3.3.2 T2 loci

T2 loci are designed to be an approximation of many T1 loci to increase performance. T2 is of particular interest when simulating long stretch of DNA. Each

T2 locus is coded as a single byte (8 bits). A T2 locus tracks the number of mutations that has happened in this locus. In other words, one can think of a T2 locus as an arbitrary number of T1 loci gathered together. If more than $2^8 = 256$ mutations have happen in a specific T2 locus, the program will send an error message and abort. There is probably much interest at creating loci which would gather more than 256 mutations but if you are interested in it, it will be easy to just change the size of each T2 locus from 1 byte to 2 bytes (or more).
Because a T2 locus represents a group to T1 loci, there are a number of selection pattern that are impossible to apply. For example, it is impossible to compute any epistatic relationship with T2 loci. It is also impossible to compute complex dominance pattern with T2 loci.
All T2 loci are initialized at caring 0 mutation at the beginning of the simulation (except eventually if the initial population is loaded form a binary file, see below `--ReadPopFromBinary`)

3.3.3 T3 loci

T3 loci are designed for quantitative genetic simulations. Each T3 locus is a single byte. The phenotype is multidimensional and each locus affect each dimension of the phenotype by a certain value set by the user. For example, consider a 3D phenotype space and a given locus which affect each of the three dimensions by 0, 0.1, 1.2 respectively. If an individual has the allele 12, then the contribution of this allele to the individual fitness is 0, 1.2, 14.4 for each dimension respectively. The user can then define a fitness landscape `--T3_FitnessLandscape (--T3_fit)`) to match a multidimensional phenotype into a single fitness value allowing for the simulation of directional selection or balancing selection. More complex fitness landscape can possibly be implemented in the code if needed.
As a T3 locus is coded as a single byte, it can take $2^8 = 256$ different values (from -128 to 127). If more precision is needed, it is easy to modify the code to allocate more memory per T3 locus. All T3 loci are initialized at the central value 0 (except eventually if the initial population is loaded form a binary file, see below `--ReadPopFromBinary`).

3.3.4 T4 loci

T4 loci are just like T1 loci except that no selection can be applied to them (and not exactly the same outputs are possible to request). Internally, however T1 loci and T4 loci are dealt with very differently. T4 loci are not directly

simulated. Instead, a coalescent tree is being recorded from the simulations and mutations are being added on to the tree whenever outputs are being requested. The only advantage of T4 loci over T1 loci results in the computational time. Use T4 loci if 1) you want many neutral loci (or the order of, say, 10^5 at least), 2) recombination rate is relatively low (of the order of, say, 10^{-7}, on average between any two locus). T4 loci are extremely fast when dealing with recombination rate of the order of 10^{-9} and lower but will be much slower than T1 loci for cases of high recombination and cases with few loci. As a general advice: don't assume that one is faster but try it out.

3.4 unif and A

In order to indicate input data in a convenient way SimBit uses a number of different Mode of input. Each specific options has its list of Modes but two Modes that come over and over again are "unif" and "A". "A" stands for "All entries" indicating to SimBit as many values as needed. "unif" stands for "uniform" on the other hand indicates that all elements will take the same value. In other words, if SimBit expect three values for a given argument, you can either do `'A 10 10 10'` or `'unif 10'`.
The usage of these keywords are a bit more complicated, typically when talking about the options `T1_fit` and `T2_fit`. See specific sections for more information.

3.5 seq, seqInt, rep and repeach

There are two special keywords `seq`, `rep` and `repeach`. When SimBit reads the input, it first tokenize the input by the option names (which start with a double dash such as `--T1_fit`) and it then evaluates the keywords `seq`, `rep` and `repeach`.

The keywords `seq` and `seqInt` are analogous to the function "seq" in R. They both expects three values the "from" value, the "to" value and the "by" value. `seqInt` is to be sued for integer values while `seq` is for (double precision) float values. For example the input `seqInt 5 17 2` can be read as "from 5 to 17 by 2" and is equivalent to `5 7 9 11 13 15 17`
The keyword `rep` is analogous to the function "rep" in R. `rep` expects two values the "whatToRepeat" value, the "howManyTimes" value. For example the input `rep 4 5` is equivalent to `4 4 4 4 4`. It is also possible to feed a vector as the first argument. Vectors must be surrounded by curly braces, values must be separated by comas. Importantly there must be no space in a vector. For example

`rep {0,0.5,1} 3` is equivalent to `0 0.5 1 0 0.5 1 0 0.5 1`
The keyword `repeach` is analogous to the function "rep" when using the `each` argument in R. `repeach` expects two values the "whatToRepeat" value, the "howManyTimes" value. `rep` and `repeach` differ only when we consider vectors. For example the inputs `rep 4 5` and `repeach 4 5` are both equivalent to `4 4 4 4 4`. For vectors things are different. While `rep {0,0.5,1} 3` is equivalent to `0 0.5 1 0 0.5 1 0 0.5 1`, `repeach {0,0.5,1} 3` is equivalent to `0 0 0 0.5 0.5 0.5 1 1 1`. While the `seq` keywords expect numbers only, the `rep` and `repeach` functions expects only the second argument to be an integer. The first argument can be any string (or a vector of any strings).

Standard ways of inputs and keywords can be mixed at will. For example `3 4 rep hello 3 0 seq 1 2 0.3 repeach {1,2,3} 3 4 4 4` is equivalent to `3 4 hello hello hello 0 1.0 1.3 1.6 1.9 3 1 1 1 2 2 2 3 3 3 4 4 4`. In older versions of SimBit a keyword `R` existed and was equivalent to the current `rep`. For backward compatibility `R` is still available but its usage is deprecated. Note also that the `R` option is available only for some options while `rep` is available for any option.

3.6 Listing all options

If you run SimBit without giving it any arguments, then it will list all the available options.

4 Species, Generation and Habitat specific options

Most options are species-specific, generation-specific and/or time-specific. SimBit uses the markers `@S` for species, `@G` for generation and `@H` for habitat. As such to refer to the, say, 120th generation one would write `@G120`.
Markers must always be indicated in order (you cannot indicate `@G150` before `@G120`). If one or several markers are missing, SimBit will assume that previous markers apply. The first marker (`@G0`, `@H0` and `@S0`) is optional. More information below.
In this section I will have to use some option that I will explain only later. Hopefully, their name will be intuitive enough to not pose too much difficulty to the reader.

4.1 Species specific options

Most options are species-specific. All species must share the same geographic location. Therefore the number of patches is not specific per patch. However, the carrying capacity can vary among species. If you want a species to be absent from a patch, just set its carrying capacity to zero. All options regarding the genetic archtecture, selection scnearios and dispersal scenarios are species specific.
You need to name (and number) your different species with the option –S (–species).

`--S (--species)name1 name2 ....` Species

Each species need a unique name. If you do not use the option `--S` SimBit, will assume you won't a single species and will simply name it "sp"!
For option that are species specific, you will refer to your species with the marker @S and by either using the species index (first species having index 0) or by the species name. For example
`--PatchNumber 1 --S Quercus Fagus --N @SQuercus A 1000 @SFagus A 60000`

or
`--PatchNumber 1 --S Quercus Fagus --N @S0 A 1000 @S1 A 60000`
are equivalent. They both ask for two species, one patch the first species having a carrying capacity of 1000 and the second species a carrying capacity of 60000.
Note that
`--PatchNumber 1 --S Quercus Fagus --N @S1 A 1000 @S0 A 60000`
`--PatchNumber 1 --S Quercus Fagus --N @SFagus A 1000 @SQuercus A 60000`

are NOT be accepted as the markers must always be given in the order in which the species names are given. If you want both species to have the same carrying capacity you can just write
`--PatchNumber 1 --S Quercus Fagus --N @S0 A 1000`
or
`--PatchNumber 1 --S Quercus Fagus --N @SQuercus A 1000`
or even
`--PatchNumber 1 --S Quercus Fagus --N A 1000`
If you want three species with the first two having same carrying capacity and the last one having a different carrying capacity you can do

```
--PatchNumber 1 --S sp0 sp1 sp2 --N @S0 A 1000 @S1 A 1000 @S2 A 200
```

or

```
--PatchNumber 1 --S sp0 sp1 sp2 --N @S0 A 1000 @S2 A 200
```

or even

```
--PatchNumber 1 --S sp0 sp1 sp2 --N A 1000 @S2 A 200
```

However, if you want `sp2` to have a different carrying capacity you can only do

```
--PatchNumber 1 --S sp0 sp1 sp2 --N @S0 A 1000 @S1 A 200 @S2 A 1000
```

or

```
--PatchNumber 1 --S sp0 sp1 sp2 --N A 1000 @S1 A 200 @S2 A 1000
```

In short, and as stated above, markers must always be indicated in order (you cannot indicate `@G150` before `@G120`). If one or several markers are missing, SimBit will assume that previous markers apply. The first marker (`@G0`, `@H0` and `@S0`) is optional.

4.2 Generation specific options

You need to indicate a priori the time for which you might want to indicate a generation-specific marker with the option `--T` or `--TemporalChanges`.

`--T (--TemporalChanges)int` Temporal Changes

You will note that `--T` is not species specific. Let's start with an example. If you require 1 patch from generation 0 to generation 1000, 5 patches from generation 1000 to generation 1200 and 1 patch again until the last generation (generation 2000) you will write

```
--T 1000 1200 --nbGenerations 2000 --PatchNumber @G0 1 @G1000 5 @G1200
 1
```

or

```
--T 1000 1200 --nbGenerations 2000 --PatchNumber 1 @G1000 5 @G1200
1
```

but you cannot write

```
--T 1000 1200 --nbGenerations 2000 --PatchNumber @G1000 5 @G0 1 @G1200
 1
```

You can also add a zero if it makes more sense to you but it is optional

```
--T 0 1000 1200 --nbGenerations 2000 --PatchNumber @G1000 5 @G0 1 @G1200
 1
```

Note that if you add extra generations in `--T`, it won't matter either (although it will eventually make the simulations negligibly slower) because the missing markers will be replaced with entries at previous markers. For example

```
--T 0 200 1000 1100 1200 --nbGenerations 2000 --PatchNumber @G0 1 @G1000
 5 @G1200 1
```

For a simulation that has one patch for 1000 generations and then 3 patches for 200 generations and finally back to 1 patch up to the end of the simulations (given by `--nbGenerations`) one would write
If you wish to have 5 patches the whole simulation you still have to indicate `@G0`. The input would be

```
--PatchNumber @G0 5
```

or simply

```
--PatchNumber 5
```

If you wish that the time-specific options to vary through time, then the specific time points at which changes must occur should be indicated by
Here is another example

```
--nbGenerations 2000 --T 0 500 1000 1200 --PatchNumber @G0 1 @G1000
 3 @G1200 1 --N @G0 unif 100 @G500 unif 1500
```

This command would indicate that temporal changes will occur at generations 0, 500, 1000 and 1200. From generation 0 to 500, there is 1 patch of 100 individual. From generation 500 to 1000, there is still one patch but with 1500 individuals. From generation 1000 to 1200 there are 3 patches of 1500 individuals each and from generation 1200 to generation 2000, there is one patch of 1500 individuals. If there is a mismatch between between arguments given to `--PatchNumber` and `--N` for any generation, SimBit will raise an error message and abort. For example the following would return an error

```
--nbGenerations 2000 --T 0 500 1000 1200 --PatchNumber @G0 1 @G1000
 3 @G1200 1 --N @G0 unif 100 @G500 A 1500
```

The reason it fails is the from generation 500 to generation 2000, a single patch

size is indicated but from generation 1000 to 1200, there are three patches and SimBit therefore does not know what the patch size should be for patch 2 and 3.

4.3 Habitat specific options

All options that relate to the selection scenario (such as `--T1_fit` for example) are habitat-specific. A habitat has to be understood in its ecological definition. Several patches may belong to the same habitat and a the habitat a given patch belong to can change over time. Patches are associated to habitat and habitat associated to specific selection scenario allowing local and temporal variation in selection. Let me explain how.
With the option `--Habitat` or `--H` one can associate any patch at any generation to a given habitat.

`--H (--Habitats)@G0 Mode int @Gx ...` Habitats

`--H` is therefore a time-specific option allowing one to change the association between patch and habitat over time and therefore to change selection pressure over both space and time. Two modes are available, `A` and `unif`. When using Mode `A`, the number of entries must be of the same length as the number of patches. Consider for example

```
--nbGens 1000 --T 500 --PatchNumber @G0 1 @G500 4 --H @G0 unif 0 @G500
 A 0 0 1 0
```

This indicates that from generation 0 to 500 there is only one patch which belongs to habitat 0. From generation 500 to 1000, there are 3 patches, the first two patches as well as the last one belong to habitat 0 and the third patch belong to habitat 1. Note that habitat 0 must always exists and it is impossible to specify an habitat index without specifying the previous one. For example it is impossible to specify habitat 5 without specifying habitats 0, 1, 2, 3 and 4. If the option `--H` is absent, SimBit assumes that all patches belong to habitat 0.

This system of associating each patch at each habitat in a time-specific manner is a very convenient solution to indicate variation in selection pressures through time and space. For all options concerning selection (`--T1_fit`, `--T2_fit` and `--T1_epistasis`), one must indicates for which patch a given selection scenario applies using the `@Hx` notation, where x is the habitat in question. You will note

the similitude between the notation `@Gx` and the notation `@Hx` for time specific and habitat specific arguments.
More information about fitness related options in the section SELECTION

5 Launching basic simulations

SimBit requires a number of basic information to make a simulation. These informations are the number of patches (`--PatchNumber`), the number of individuals per patch (`--N` or `--PatchCapacity`), the number of loci, their types and physical ordering on the chromosome (–L or –Loci), the number of generations (`--nbGens` or `--nbGenerations`), the mutation rate (`--T1_mu` or `--T1_MutationRate` for T1 loci and `--T2_mu` or `--T2_MutationRate` for T2 loci), the recombination rate (`--r` or `--RecombinationRate`; is required only if there is more than one locus indicated by option `--L`) and the dispersal rate (`--m` or `--DispMat`; only required if there is more than one patch). Note that I call different panmictic patch in a structured population, "patch" and not "population", "subpopulation" or "deme".

For a start we will consider the following simple example

```
./SimBit --nbGens 5000 --PatchNumber 4 --N A 100 100 100 100 --m Island
 0.01 --L T1 3 --T1_mu A 0.00001 1e-8 1e-8 --r rate A 1e-6 1e-6
```

This simulation should run in a few seconds (When copy-pasting code from the manual to the terminal, be aware that depending on the position of a newline, there might have issue. If there is any issue you might want to first copy on your favourite text editor and then onto the terminal). Note that this exact same command can be found in the file `exampleCommand`. To run this simulation when reading the command form the file, use

```
./SimBit file exampleCommand.txt 1
```

The number of `1` at the end indicate that SimBit must read the first line from the file. This allows a user to gather a number of different commands in the same file.

The above command probably makes little sense to you so far so let's go through it. The command asks for a simulation lasting 5000 generations with 4 patches

of 100 individuals each. Migration follows a classical island Model in which the probability of NOT migrating is 0.95. Each haplotpye is made of 3 loci of type T1. The mutation rate per locus is 0.00001 ($1e^{-5}$), $1e^{-8}$ and $1e^{-8}$ respectively. The recombination rate between any locus and the next is $1e^{-6}$.

The first option is quite straightforward, the argument of `--nbGeneration` is a single integer number

`--nbGens (--nbGenerations)int` — Number of generations

The second option here is `--PatchNumber`. This option indicates the time-specific number of patches

`--PatchNumber @G0 int @Gx ...` — Number of patches

The third option is `--N`. This option indicates the time-specific number of individuals per patch. Above I used `--N @G0 A 500`. "A" is the 'Mode' used to input data. "A" stands for "All entries". For `--N`, there are two possible Modes "A" and "unif".

`--N (--PatchCapacity)@S0 @G0 Mode int ...` — Patch carrying capacity

Because all patches have the same patch size in the above example (`--N A 500 500 500 500`), it would be equivalent to input `--N 'unif 500'`. When choosing the Mode "A", the number of values to be inputted must be equal to the PatchNumber for this specific range of generations. Note that in the current version, the carrying capacity of a patch is not changing with any selection pressure.

Now comes option `--m`. This options specifies the time-specific dispersal scenario. Data can be inputted with different Modes `A`, `LSS`, `OnePatch`, `Island`, `LinearNormal`. `A` is the most flexible mode of entry and expect the user to input the entire matrix of dispersal probabilities. More information about these different modes in table 1. By default `--m` is set to OnePatch.

`--m (--DispMat)@S0 @G0 Mode value ...` — Migration

The next option is `--L`. This options specifies the number, types and ordering of loci.

`--L (--Loci)@S0 LocusType NbLoci LocusType NbLoci LocusType NbLoci ...` Loci

For example, if you want 23 T2 loci, followed by 1000 T1 loci followed by 12 T2 loci, followed by 50 T3 loci you could input

```
--L 'T2 23 T1 1e5 T2 12 T3 50'
```

Note that `--L 'T2 23 T1 5000 T1 5000 T2 3 T2 9 T3 10 T3 40'` would have the same effect. For backward compatibility the letter `T` in input to the option `--Loci (--L)` is optional. `--L 1 3` is equivalent to `--L T1 3` (you can also write `--L t1 3`). The recombination rate between any of these loci, including, the placing of loci on independent chromosomes is indicated via the option `--r (--RecombinationRate)`. For the above case, the option `--r` will expect 1034 entries as there are 1035 loci (see below).

The next option is `--T1_mu`. This option specifies the mutation rate for each T1 locus. There are two Modes of input `A` and `unif`

`--T1_mu (--T1_MutationRate)@S0 Mode float ...` Mutation Rate on T1 loci

When using the Mode `A`, the number of entries must equal the number of T1 loci indicated by `--L`. Below is an example.

```
--T1_mu A 1e-8 1e-7 1e-7 1e-7 1e-7 1e-7 1e-9 1e-8 1e-8 1e-8
```

Using the `rep` keyword, this entry could be rewritten as

```
--T1_mu A 1e-8 rep 1e-7 5 1e-9 rep 1e-8 3
```

The options `--T2_mu`, `--T3_mu` and `--T4_mu` are not present in the above code but I will mention it here for their similitude with `--T1_mu`. `--T2_mu`, `--T3_mu` and `--T4_mu` specifies the mutation rates for T2 loci, T3 loci and T4 loci respec-

Table 1: Inputing data for –m

Mode	Meaning	Example	Default
A	Input a PatchNumber x PatchNumber square matrix with the probability of migration from each population to any other population. The values of such dispersal matrix are indicated with patch of origin listed downwards (i.e. row names), and patch of destination listed horizontally (i.e. column names), and that the matrix is then listed by going through rows first (i.e. left to right, then top to bottom)	'A 0.8 0.2 0 0 0.1 0.8 0.1 0 0 0.1 0.8 0.1 0 0 0.2 0.8' would simulate a 4 patches stepping stone model	No
LSS	It stands for 1D Linear Stepping Stones. Here the first element is the number of probabilities to expect next. Then is a vector of probabilities and finally is which element of this vector (zero based counting) corresponds to the probability of not migrating. The resulting dispersal matrix is corrected at the edge (reflective boundary effect).	'LSS NbProbabilities probabilities center'. For example 'LSS 4 0.05 0.05 0.75 0.15 2' indicates that the probability of not migrating is 0.75, the probability of migrating one patch on the left is 0.05, the probability of migrating two patches on the left is 0.05, the probability of migrating one patch on the right is 0.15.	No
OnePatch	It is the only possible entry when there is only a single patch	'OnePatch'	
Island	This creates a classical island Model. One value is expected which is the probability of migrating	'Island ProbabilityOfMigrating' For example 'Island 0.01' would create an island Model where the probability of migrating from any patch to any other patch is 0.01.	Yes
LinearNormal	This creates a 1D dispersal kernel that is approximated by a normal (guassian) function. The two entries expected are the standard deviation of this Gaussian distribution (in number of patch) and the number of standard deviation above and below which the probability of migrating will be approximated to zero	'LinearNormal SD nbSD' For example 'LinearNormal 2 4' indicates a 1D gaussian distribution kernel with 2 patches of standard deviation and that after 4 standard deviation, the probability of migrating will be considered sufficiently low to be approximated to 0.	No

tively and work exactly as `T1_mu` works

`--T2_mu (--T2_MutationRate)@S0 Mode float ...` Mutation rate on T2 loci

`--T3_mu (--T3_MutationRate)@S0 Mode float ...` Mutation rate on T3 loci

`--T4_mu (--T4_MutationRate)@S0 Mode float ...` Mutation rate on T4 loci

The last option in the above example is `--r`. This options specifies the recombination rate between any two loci (whether the two loci in question are T1 or T2 loci). The first element to input the the Unit of genetic distance used. There are 3 possible Units; `rate`, `cM` and `M`. `rate` means that values represent rate of recombination. `cM` indicates that values represent centiMorgans. `M` indicates that values represent Morgans (1 Morgan = 100 centiMorgans). Of course, there is not much difference between `M` and `rate` if distances are not too high (say below 0.1).

Again there are two Modes, `A` and `unif`. For the Mode `A`, the keyword `R` (explained above) is available and the number of entries should equal the total number of loci minus 1, where the total number of loci equals the number of T1 loci plus the number of T2 loci. The above `--r 'rate A 1e-6 1e-6'` could also be written as `--r 'rate A rep 1e-6 2'` or `--r 'rate unif 1e-6'`. Perfect independence has to be indicated with '-1' if Unit is `cM` or `M` and can be indicated as '0.5' or '-1' if Unit is `rate`.

`--r (--RecombinationRate)@S0 Unit Mode float ...` Recombination

The option `--r (--RecombinationRate)` is the only option that requires the input of a unit. s like it does in a Wright-Fisher population, that is at a rate of 1/2N (a little lower in fact when there is migration).

6 Selection and phenotype

Selection scenarios are quite flexible in SimBit. In absence of epistasis (as indicated by option `--T1_epistasis`), fitness effects among loci are multiplicative. As calculating epistatic interactions is slow in comparison to other fitness calculation, it would of interest to some to have fitness effects that are additive among loci. If it is the case, please let Remi Matthey-Doret know and he can probably help you to implement that.

For whatever selection scenario you suggest, SimBit will search for the fastest way to make the simulation. There are three options here, all of them are habitat specific. As a reminder from section HABITAT SPECIFIC OPTIONS, if there is no need for local selection, then there is no need to use the option `--H` (or `--Habitats`) and SimBit assumes that all patches belong to habitat 0. Therefore the flag `@H0` is always required when giving arguments to selection related options.

`--T1_fit (--T1_FitnessEffects)@S0 @H0 Mode float @Hx ..` Selection on T1 loci

There are five possible Modes; `A`, `domA`, `unif`, `MultiplicityA`, `MultiplicityUnif` , `MultiplicityAGamma`. All Modes starting by "Multiplicity" means that you are willing to assume "multiplicity of dominance effects" that is genotype 0|0 has a fitness of 1, genotype 0|1 has a fitness 'x' given in input and genotype 1|1 has a fitness x^2 (hence the term multiplicity). The selection is therefore necessarily against the allele 1. Making this assumption allows to considerably improve SimBit performance (for several reasons not detailed here). In all Mode except `MultiplicityAGamma` entries must be fitness values and not selection coefficients. In other word, if a mutation has a selection coefficient of 0.1, one must input the value 0.9. More information in table 2. We highly recommend using the options starting with "Multiplicity" as it will greatly improve the runtime.

Note that in the current version, it is impossible to make the multiplicity assumption for a given habitat but not for another one. It will raise an error message. This limitation could be eliminated if you really need to, please contact Remi Matthey-Doret for help.

For T2 loci the logic is very similar

Table 2: Input of data for –T1_fit

Mode	Meaning	Example	Number of entries
A	Indicates the fitness of all three genotypes at all loci	`A Locus1_G0|0 Locus1_G0|1 L1G1 |1 Locus2_G0|0 Locus2_G0|1 Locus2_G1|1`	3 * number of T1 loci
domA	It is just like 'A' but allow to reduce the size of the input (and the computation on the user side) when you want to assume that the homozygote wildtype has a fitness of 1.0. First indicate 'cst' of 'fun' followed by a float number indicating the average dominance coefficient $\bar{h}$. if 'cst' is used, then all loci have the same dominant coefficient $\bar{h}$. If 'fun' is used, then the dominance coefficient at all loci is given by $h_i = \frac{e^{-ks_i}}{2}$, where $k = \frac{-log(2\bar{h})}{\bar{s}}$, where $\bar{s}$ is the average selection coefficient.	`domA fun 0.2 Locus1_G011 Locus2_G1|1 Locus3_G1|1`	number of T1 loci + cst or fun + $\bar{h}$
unif	Indicates the fitness components for all three genotype and assume that all loci are under the same selection scenario	`unif G0|0 G0|1 G1|1`	3
MultiplicityA	Indicates the fitness components of the genotype 0|1 at each locus separately and makes the assumption of "multiplicative dominance effect"	`MultiplicityA Locus1 Locus2 Locus3`	Number of T1 loci
MultiplicityUnif	Indicates the fitness components of the genotype 0|1 for all loci makes the assumption of "multiplicative dominance effect"	`MultiplicityUnif G0|1`	1
MultiplicityAGamma	Just like `MultiplicityA` except that the fitness values are 1 minus the selection coefficient which are drawn from a gamma distribution with parameters alpha and beta given by the user	`MultiplicityAGamma alpha beta9`	2

`--T2_fit (--T2_FitnessEffects)@S0 @H0 Mode float ...` Selection on T2 loci

Here there are only three Modes: A, unif and gamma. In all cases, it assumes multiplicity of dominance effects. (One may argue that the Modes might better be renamed MultiplicityA, MultiplicityUnif and MultiplicityGamma).

For epistatic interactions use

`--T1_epistasis (--T1_EpistaticFitnessEffects)@S0 @H0 nbGroupsOfLoci` Epistasis
`nbLociUnderEpistasisFirstGroup`
`LociIndices fitnesses nbLociUnderEpistasisSecondGroup LociIndices fitnesses`
`...`

Note that `--T1_epistasis` is completely independent of `--T_fit`. As such a locus could be under both non-epistatic selection and epistatic selection. The effects would be multiplicative. It is up to the user to decide whether (s)he want such thing or not. If this is the case SimBit shall throw a warning message though.

The user must indicate for each species and habitat the number of groups of loci to be considered. Within each group, the user must indicate the number n (above called `nbLociUnderEpistasisFirstGroup` and `nbLociUnderEpistasisSecondGroup`) loci under epistasis. With n loci, there are n loci indices and 3^n fitness values to indicate. Let's take the example of having 3 loci under epistasis, we have to indicate $3^3 = 27$ fitness values. Let's call the three loci A, B and C and three possible genotypes at each locus 0|0, 0|1 and 1|1. Let's write the fitness of the genotype 0|0 at locus A, 0|1 at locus B and 0|1 at locus C by `A0|0_B0|1_B0|1`. Let `IA`, `IB`, `IC` be the indices of each of these three loci. Please note that the first locus has index 0 and the last locus has index "Number_T1_Loci - 1". One would write

```
--T1_epistasis 3 AI BI CI A0|0_B0|0_C0|0 A0|0_B0|0_C0|1 A0|0_B0|0_C1
|1 A0|0_B0|1_C0|0 A0|0_B0|1_C0|1 A0|0_B0|1_C1|1 A0|0_B1|1_C0|0 A0|0
_B1|1_C0|1 A0|0_B1|1_C1|1 A0|1_B0|0_C0|0 A0|1_B0|0_C0|1 A0|1_B0|0_C1
|1 A0|1_B0|1_C0|0 A0|1_B0|1_C0|1 A0|1_B0|1_C1|1 A0|1_B1|1_C0|0 A0|1
_B1|1_C0|1 A0|1_B1|1_C1|1 A1|1_B0|0_C0|0 A1|1_B0|0_C0|1 A1|1_B0|0_C1
|1 A1|1_B0|1_C0|0 A1|1_B0|1_C0|1 A1|1_B0|1_C1|1 A1|1_B1|1_C0|0 A1|1
```

`_B1|1_C0|1 A1|1_B1|1_C1|1`

For T3 loci, one must indicate how the genotypes match to phenotypes with

`--T3_pheno (--T3_PhenotypicEffects)@S0 int @H0 Mode float @H0 Mode float @Hx ... @Sx int @H0 ...`

T3 phenotype and plasticity

The input format is a bit unusual here as this option expects an integer value and then habitat-specific arguments each with a mode a float numbers. The integer value is the number of dimensions of the phenotypic space. There are two modes, 'A' and 'unif'. For Mode 'unif', the expected number of entries is the number of dimensions of the phenotype and all loci will have the same impact on the phenotype. For Mode 'A', the expected number of entries is the number of dimensions times the number of T3 loci. For example

`--Loci 3 2 --T3_pheno 3 A 0 0 0.5 1.1 0.1 0.2`

indicates a case where there are 2 T3 loci, the first locus affects only the last dimension of the phenotypic space and the second locus affects all dimensions. If the first locus has value -5 and the second locus has value 10, then the contribution to the phenotype for this specific haplotype (which will be added to the contribution of the other haplotype) is 11 1 -0.5. The phenotypic value along the ith, Z_i is therefore

$$Z_i = \sum_{j=0}^{j=L} loc_{j,i} \cdot (A_{1,j} + A_{2,j})$$

, where L is the number of loci, $loc_{j,i}$

is the effect of the jth locus on the ith phenotypic axis and $A_{1,j}$ and $A_{2,j}$ are the allelic values at the jth locus of the alleles on the first and second haplotype respectively.

Because `--T3_pheno` is a habitat-specific variable, one can model phenotypic plasticity.
The option `--T3_fit (--T3_FitnessLandscape)` allows to match a given phenotype to a fitness value

`--T3_fit (--T3_FitnessLandscape)'SelectionMode @H0 EntryMode mean gradient/omega @Hx ...'`

Selection on phenotype (T3)

The 'SelectionMode' can be 'linear' or 'gauss'. If 'SelectionMode' is 'linear', then the fitness component of a given phenotypic axis is calculated as a linear regression with 'mean' and 'gradient' (or slope) given after the EntryMode. Let D be the number of dimensionsz_i be the phenotypic value along the ith axis, $z_{opt,i}$ be the optimal phenotype and g_i be the gradient (or slope) along this same axis, then the fitness is defined by

$$W = \prod_{i=1}^{i=D} 1 - |z_i - z_{opt,i}| \cdot g_i$$

, where $|x|$ means absolute value of x. If for any dimension i the quantity $1 - |z_i - z_{opt,i}| \cdot g_i$ is zero or negative, then of course, the fitness is set to 0.

If 'SelectionMode' is 'gauss', then fitness is given by a gaussian function with mean and selectionStrength omega. Fitness is then given as

$$W = \prod_{i=1}^{i=D} exp\left(-\frac{(z_i - z_{opt,i})^2}{\omega}\right)$$

The two possible 'EntryMode' are 'A' and 'unif'. For 'unif', 2 values are expected (`unif mean gradient` or `unif mean omega`). For 'A', $2 \cdot D$ values are expected (`A mean_1 gradient_1 mean_2 gradient_2 ...`).

7 Fecundity and patch size

One will note that I use the term patch size to refer to the number of individuals in a patch at a given generation, and the term patch carrying capacity to refer to a maximal limit of the patch size (imposed by option `--N (--patchCapacity)`). `--fec (--fecundityForFitnessOfOne)@S0 float ...`

fecundity

By default, this number is set to -1. In such case, the patch size is always at carrying capacity. You could set the fecundity to an arbitrarily large value to obtain the same effect as -1 but that would (very slightly) slow down the simulation. The fecundity is understood as a per individual measure. Take note

that when using males and females, only the fecundity of females will affect the number of offspring produced (as long as there is at least one male in the patch). If we assume all fitnesses are at 1, then if we have hermaphroidtes a fecundity of at least 1 will be necessary to replenish the entire patch of individuals (there is stochastic variation that will cause on average a decrease in the patch size). If you use a males and females mating system with a sex ratio of say, 0.75 (3 male for 1 female), then you will need a fecundity of at least 4. The actual number of individuals in the population at the next generation is sampled from a Poisson distribution leading to some stochasticity in the patch size. The patch size is truncated at carrying capacity and can never exceed it. For fluctuations in population size around a carrying capacity use option `--growthk` explained below.

The fecundity will interact with the effect of ecological relationships with other species. See section "Multiple Species"

When dealing with a fecundity that is not infinite, the patch size may differ from the carrying capacity. Part of why SimBit is so fast is due to the way it allocates its memory and as a consequence SimBit will keep a hard limit on the carrying capacity with no way to overshoot it. However, you can limit a logistic growth with potential overshoot to another carrying capacity. This is where the option `--growthk` comes into play.

logistic growth curve

`--growthk @S0 @G0 Mode float`

There are two possible modes, `unif` and `A`. In the `A`, you have to indicate the carrying capacity fora logistic growth for each patch. SimBit will not enforce the fact that the value entered is lower that the hard coded carrying capacity indicated with option `--N (--patchCapacity)` as some users may want for some specific interest make this kind of simulations. For those of you that are used to see matrix of species interaction, the values present in the diagonal of a matrix of interaction are $-\frac{-1}{growthK}$. The difference in SimBit is that SimBit allows this value to be set differentially in each patch.

8 Mating system

Cloning Rate

`--cloningRate @S0 float ...`

The rate at which cloning happen. Note while cloning, mutations are still happening so that offspring might not be exactly a clone of its parent

`--selfingRate @S0 float ...`

Selfing Rate

The rate at which selfing happen. When using a cloning rate different from zero, the selfing rate is the selfing rate for offsprings that are not produced via cloning. A selfing rate of 0.0 (default value; or -1.0) means that selfing rate occurs just like it does in a Wright-Fisher population, that is at frequency 1/N.
Note that cloningRate has preceedence over selfingRate. This means that if you use both options, then the selfing rate will be conditional on no cloning happened. For example, if you set the cloningRate to 1, then no selfing (or any other sort of sexual reproduction) will ever occur. If you set the cloningRate to 0.5 and the selfing rate to 0.1, then 50% of the offsprings will be made through cloning, 5% through selfing (because 10% of 50% is 5%) and the other 45% through normal reproduction.

Mating System

To set whether it runs hermaphrodites or males and females, use `--matingSystem @S0 system (sexRatio)...`
There are two possible mating systems. 'h' (or 'H') for hermaphrodites. This is the default. And 'fm' (or 'mf', 'FM,'MF') for males and females. If you specify males and females, you will then need to specify the sex ratio (the ratio of males) in the population (e.g. `--matingSystem fm 0.5` to set all species to a males and females mating system with an even sex ratio).

9 Multiple Species

Most options are species specific (such as `--L`, `--m`, `--H`, `--T1_fit`, ...) and the goal of this section is not to review them again. This section is here to present ecological relationships and variation in generation time among species.

Species Ecological Relationships

`--eco (--ecoRelation, --SpeciesEcologicalRelationships) type float type float type float ...`

The option expects a matrix of interaction between any two species present. Each interaction between two species is characterized by a 'type of interaction' and an 'effect' (or 'magnitude'). The three types are 'A', 'B', 'C' and '0' (see table below). An effect of 0.0 means, no effect. The diagonal of the input matrix (the effect of each species on itself) is necessarily made of zero '0 0.0' and you must not indicate them. Note that the effect of species A on species B is not necessarily the opposite of the effect of species B on species A. It is important to understand how this effect is affected by the patch size of both species as given by the different types.

'A' means 'effect is multiplied by the patchSize of the causal species'
'B' means 'effect is multiplied by the patchSize of the causal species and divided by the patchSize of the recipient species'
'C' means 'effect is divided by the patchSize of the recipient species'
'C' means 'effect is multiplied by both the patchSize of the recipient species and of the causal species'
'0' means no effect. It must therefore necessarily be followed by an effect (or magnitude) of 0.0.

The effect is added on the fecundity of individuals of the recipient species. Consider a case where a 'focal' species has one 'predator' and one 'prey' in the environment. Their matrix of interaction could be

```
--S prey predator focal --eco 0 0.0 B 2 0 0.0 A -4 A -5 B 2.2
```

Here I assumed the prey and predator species do not affect each other. I also assumed the predatory effect were causing a fixed number of death based on the number of predators but did not depend upon the number of preys. On the other hand, I assumed the beneficial effect of preying on another species depends on both the patch size of the prey and the predator.

It is also possible to vary generation time between species. This is achieved with

```
--nbSubGens (--nbSubGenerations)@S0 int @S1 int ...
```

Generation time

A "SubGeneration" is a generation within a generation. This allows you to simulate a species that, say 4 generations, every time another has one generation. It is impossible to have a species with a generation time 1.2 times faster than another unfortunately as the number of "SubGeneration" per generation must be an integer. For example

```
--nbSubGens (--nbSubGenerations)@S0 1 @S1 2
```

would cause species 0 to have a generation that twice the one of species 1. It

would make little sense to input something like

```
--nbSubGens (--nbSubGenerations)@S0 2 @S1 4
```

Each species would undergo a number of "SubGeneration" per generation but it would probably be easier to just double the number of generation given to `--nbGens (--nbGenerations)`. Of course, by default all species only have a single "SubGeneration".

10 Initial Population

By default All T2 loci are set to carrying 0 mutations and all T1 loci are set to 0. It is possible however to indicate the per patch allele frequency with `--T1_Initial_AlleleFreqs`

`--T1_Initial_AlleleFreqs 'Mode (value)'` — T1 Initial allele frequencies

There are four Modes; `AllOnes`, `AllZeros`, `A`, `Shift`. See table 3 for more information. The parenthesis around "value" (`value`) above indicates that the input of values depend on the Mode. `--T1_Initial_AlleleFreqs` is somewhat limited by its flexibility. For a more complex description of the starting population, it is possible to use the option `--resetGenetics` at the first generation.

Another option is to directly import a population saved in a binary file from a previous simulation. For this use

`--ReadPopFromBinary 'file'` — Import population

In order to read a binary file correctly, one must know what it is reading. For this reason it is essential to specify correctly the number of type of loci, number of patches and number of individuals per patch with the usual `--PatchNumber --N` and `--L` options.

`--InitialpatchSize ints` — Initial patch Size

Set the initial patch size for each patch. The expected number of values should

Table 3: Input data for –T1_Initial_AlleleFreqs

Mode	Meaning	Example	Default
AllOnes	Set all T1 loci to 0	`AllOnes`	Yes
AllZeros	Set all T1 loci to 0	`AllZeros`	No
A	Specify the per patch, per T1 locus allele frequency. Note that all loci would then have the same allele frequency. There is currently not option to set the per locus allele frequency but it would be easy to implement so please just contact Remi Matthey-Doret if you are interested in such feature	`A AlleleFreqPatch1 AlleleFreqPatch2 AlleleFreqPatch3 ...`	No
Shift	Specify a given patch before which allele frequencies (at all T1 loci) are set to 0 and after which allele frequencies (at ll T1 loci) are set to 1	`'Shift PatchWhereTheShiftOccurs'`. For example `'Shift 4'` indicates that for patches "0, 1, 2 and 3" allele frequencies are set to 0 and for patches "4, 5, 6, ..., PatchNumber-1" allele frequencies are set to 1. Please note that first patch is the patch 0.	No

equal the number of patches at the start of the simulation and no initial patch size should be be larger than the initial carrying capacity.

11 Reset genetics

This option deserves its own section.

`--resetGenetics @S0 event <generation> <TraitType> <typeOfMutationsIfNeeded> <lociListInformation> <haplotypesInformation > <patchAndIndividualInformation> event etc... @S1 etc...` Reset genetics

This option allows a user to make "artificial modification" to a simulation while it is running. It allows for example to add a mutation to simulate a selective sweep, or to reset an entire population to the wild type allele. Note that it does not allow one to change the genetic architecture (number or type of loci) but only the mutations that the different individuals are carrying. The option will take a number of events. An event is a set of actions that SimBit will take to affect the genetics of a population. Every event must start with the keyword 'event' (without quotation mark).

Directly after the keyword "event" comes the trait type to be affected in this event (T1, T2 or T3). If the trait type is 'T2' or 'T3', then SimBit will just reset the designated loci/individuals/patch to zero. If the trait type is T1, then an extra specification must be given to describe the type of mutations. There are three possible types of mutations 'setTo0', 'setTo1' and 'toggle'.

It is followed by loci list information. The user can either say "allLoci" and all loci will be affected or list the loci with the keyword "lociList". For example to affect loci indices 0, 4 and 7, indicate "lociList 0 4 7".

Afterward comes the haplotypes information. It must start with the keyword "haplo" and be followed by either '0', '1' or 'both' to indicate on which set of chromosome (individuals are diploid as a reminder) the mutation will happen.

Finally comes the patch and individual information. It must be formatted as `patch 0 <individuals information> patch 4 <individuals information>` . To affect all individuals, use the keyword 'allInds' and to affect only specific

individuals use the keyword 'indsList' (e.g. `indsList 1 2 3 4 5`). For example to affect all individuals of patch 0 and only individuals 10, 20 and 25 of patch 3, you would write `patch 0 allInds patch 3 10 20 25`.

Let's do a full example. Let's assume we would like to simulation a selective sweep. We want only one mutation on the 11th T1 locus (index 10 by zero based counting) of the first (index 0) individual in the first (index 0) patch to happen at generation 100. We could do

```
--resetGenetics event 100 T1 setTo1 lociList 10 haplo 0 patch 0 indList
 0
```

Of course, if this specific individual, on this specific allele was already mutated, nothing would happen as the mutation type was 'setTo1'.
Let's consider another example. Let's say we want to reset the loci 10 15 and 20 to be fixed for allele 0 in one patch and fixed for allele 1 in the other patch. We would need to events for that.

```
--resetGenetics event 500 T1 setTo0 lociList 10 15 20 haplo both patch
 0 allInds event 500 T1 setTo1 lociList 10 15 20 haplo both patch 1
 allInds
```

Note that the genetic reset will happen at the end of the specified generation but just before writing the outputs for this generation. So, in the above example, the offsprings of the generation 500 (that is the parents of the generation 501) are going to be reciprocaly fixed at loci 10, 15 and 20.

If the fecundity (–fec) is not set to -1, it is possible that the patch size differs from the carrying capacity. If an individual does not exist, SimBit will of course not try to mutate this individual. If you want to simulate a single mutation, it is therefore more strategic to indicate a small individual index.

12 Advanced dispersal options

In section, Launching Basic Simulations, I already described the basic –m (–DispMat) option. I won't talk about it again here. I just want to add a few more options that relates to dispersal.

Selection softness

`--DispWeightByFitness 'int'`

`--DispWeightByFitness` allows to specify whether the migration rate should be weighted by the mean fitness of the emigrant patch. If the migration rate is weighted by fitness, it would simulate a case of hard selection, otherwise it would be a case of soft selection. Note that when the fecundity differs from -1, then dispersal is necessarily weighted by fitness. Two entries are possible `0` and `1`. `0` is default and means soft selection and `1` means hard selection. Choosing `0` might make the simulation a little bit faster especially for simulations with a lot of generations.

gamete dispersal

`--gameteDispersal @S0 yes/no ...`

If the option 'gameteDispersal' is set to 'yes', then gametes disperse and the offspring will leave wherever the gametes meet. If 'no', then the offspring migrate. Concretely, if gamete disperse, the two parents can be from different patch. If offpsring disperse, then the two parents must be from the same patch. For most case, the difference between the two won't matter. Because offspring dispersal is computationally slightly faster (although often negligibly so), the default is 'no'.

13 Technical options and performance

Random seed

`--random_seed (--seed)'Int'`

`--random_seed` specifies the random seed for the simulation. The seed will be printed on the logfile if this info has been demanded. Knowing the random seed allows one to replicate the exact same simulation which is often very handy. Instead of specifying an `Int`, you can also use the keyword 'binfile' or 'f' and give the path to a binary file containing the seed. Such binary files can be produced from other simulations and outputted thanks to

By default, the random seed is set on the computer current time.

It is sometimes helpful to start a simulation at a generation other than 0. This is typically useful when restarting a simulation (from a binary file) who crashed because of overpassing the walltime limit on a cluster). In such case, you can use

Table 4: Input data for –Overwrite

Entry	Meaning	Default
0	Do not overwrite	No
1	Overwrite even if the logfile already exists but not if the last output files exist	Yes
2	Overwrite in any case	No

`--startAtGeneration --startAtGeneration 'Int'` Generation to start from

By default, `startAtGeneration` is set to 0.

`--Overwrite 'Int'` Overwrite

`--Overwrite` can take values 0, 1 or 2.

By default Overwrite is set to 1.

`--DryRun 'Int'` DryRun

To ask for a DryRun indicate `--DryRun '1'`. In such case SimBit will do everything as normal but will exit just because running the first generation.

Just like any software, SimBit has its strengths and weakness when it comes to performance. SimBit is extremely fast when it comes to T1 loci when performing the assumption of multiplicity (`MultiplicityUnif`, `MultiplicityA`) or T2 loci (T2 loci necessarily assume that dominance effect are multiplicative). If you are willing to assume that indeed the fitness effect for the three possible genotypes at a given locus are 1, $1-s$ and $(1-s)^2$, then SimBit can remember the fitness effects of individual sequences and won't need to recompute the fitness unless a mutation or a recombination event occurred in this sequence. SimBit therefore partition the genome to give each sequence the same probability of an event to occur that would force SimBit to recompute the fitness for this sequence. The probability assigned to each of these sequence if given by the option `--FitnessMapInfo` . By default, the probability is set to 0.001 (`--FitnessMapInfo prob 0.008`). It is a value that generally speaking works pretty well but you might want to tweak this number to optimize running time for your particular simulations. `--FitnessMapInfo` use to take several modes but it was reduced to the mode that

worked best `prob`.

`--FitnessMapInfo @S0 Mode float @S1 Mode float ...` T1 Performance tweak 1

`--FitnessMapInfo` is an advanced option that serves no purpose but trying to make your simulations faster but it is not going to change anything to what is being simulated. You can try to tweak it (independently for each species) to make your simulation faster if you use the assumption of multiplicity or T2 loci. The only possible Mode is `prob`. To make a long story short, the default is a bit more complex that what the user can input but it is something along the lines of prob 0.008. Changing this option will only be effective if you use the "multiplicity assumption" under `--T1_fit` and/or if you T2 loci with selection (`--T2_fit`).

`--resetTrackedT1Muts @S0 int @S1 int ...` T1 Performance tweak 2

For performance reasons, SimBit can compute fitness for T1 loci by only looking at "loci of interest". I call "loci of interest", here, the loci that are 1) either are fixed for an allele that might affect fitness in a given habitat or 2) are polymorphic and might affect the fitness. This allows to drastically boost the time to compute fitness, esp. noticeable when not using the multiplicity assumption. The drawback is that it means that SimBit must keep track of the loci of interest and doing so is computationally expensive. At the end, it is a complicated trade-off between these two processes. When the option –resetTrackedT1Muts is set to -1, then SimBit will just compute fitness over all sites and not keep track of the "loci of interest". If a strictly positive value, n, is inputted, SimBit will keep track of loci of interest and will screen the population to remove the loci that are no longer of interest every n generations.

By default, n is set to -1 if the assumption of multiplicity is asked for. If the user does not ask for the assumption of multiplicity and if the expected fraction of polymorphic sites is not too high, then n is set to 10 generations. While SimBit tend to do a pretty good job at estimating the `--fitnessMapInfo`, default parameter, it appears quite harder to estimate a good `--resetTrackedT1Muts` parameter. Hence, you might want to try out different values to minimize the computational time. Note that The computational time will depends upon the number of poly

I recommend to try out different inputs for `--FitnessMapInfo` if you use T1 with multiplicity and/or T2 loci and try out different inputs for `--resetTrackedT1Muts` when using T1 loci (esp. when not using the multiplicity assumption but not only) to try to tweak their simulation to make them as fast as possible!

T4 loci are not directly simulated. Instead a coalescent tree is being computed and mutations are added on the coalescent tree. In presence of recombination, every locus can potentially have its own evolutionary history. Instead of representing every lineage, SimBit record some ancestral recombination graph (ARG) inspired by Keheller et al. (2018). At any time, a given haplotype can be described by a number of nodes in the ARG. As the average number of nodes per haplotypes increase, the simulations get slower. It is in general not too much of an issue, as drift is often sufficient to avoid that this average number of nodes per haplotypes to reach to a high value. That being said, it can become an issue. For this, SimBit can redefine the ancestral nodes and clear the tree on demand when the average number of nodes per haplotype is too high. The threshold for such action to be taken is given through the option.

T4 loci performance tweak

`--T4_maxAverageNbNodesPerHaplotype @S1 float @S2 ...`

By default this option is set to 100.

14 Outputs

General Path

`--GP (--GeneralPath)path`

The option `--GeneralPath` indicates the path where all the outputs will be printed. Other paths relative to outputs are all relative to the "GeneralPath". SimBit does not add a terminal "/" to the path. So, if your path does not end with a "/", then the characters after the last "/" are taken as a prefix of all output files. There is no default for this option, so you have to input at least an empty string if you wish any outputs, otherwise an error will be thrown.

In all of below outputs you have toinput a filename that comes after the "GeneralPath". There are two important keywords; "nfn" and "NFN", which stand for "No FileName". If you indicate "nfn" (or "NFN") as filename for a specific

output, then the file will have have a specific name (but only a specific extension and generation specific information or other specific information). This is handy because it allows the user to give a standard name for files directly in the "GeneralPath". For example

```
--GP /path/to/directory/firstSimulation -- T1_vcf_file nfn 200 300
400 500 --T1_AlleleFreq_file nfn 500
```

will create the same output as

```
--GP /path/to/directory/ -- T1_vcf_file firstSimulation 200 300 400
 500 --T1_AlleleFreq_file firstSimulation 500
```

There are 3 general classes of outputs; the Logfile, a binary file of the population and various user friendly (tab separated values and VCF) outputs.
Each input requires first a file name and then, if applicable, timing of when the output must be produced. As a general advice: avoid using spaces in the names of files or directory and this might eventually be misinterpreted. Outputs that are species specific are automatically produced for each species and the file name is then preceeded by the species name.
For all the outputs provided, you can ask for a version of these outputs containing sequencing errors.

`--sequencingErrorRate rate`

Sequencing error

Using this option with a rate different from 0.0 will cause SimBit to produce all outputs as asked (original data without sequencing error) plus an extra set of outputs with simulated sequencing errors. The files with sequencing error will have the string "_sequencingError" added to the file name just before the extension.

14.1 Logfile

The 'logfile' can be used to 1) remember what input has been given to SimBit and 2) to make sure SimBit interpreted the input as we wished although this second usage might not be for beginners. For all outputs that can be

Entry	Meaning	Default	Comment
0	No Logfile is being printed	No	-
1	Logfile contains only the arguments that SimBit has received through the command line	Yes	-
2	Logfile contains the arguments that SimBit has received through the command line and all the parameters that has been set for the simulation	No	logfile might be very big

`--Logfile 'filename'` Logfile

SimBit will automatically add the extension '.log' to your filename. By default, the filename is 'logfile.log'.

It is possible to specify the type of logfile we want

`--LogfileType int` Logfile Type

Three possible entries are possible; 0,1 and 2
For all file output for which generations at which we need the outputs must be indicated, one can either write out every generations (e.g. `--T1_vcf_file filename 10 20 30 40 50 60 70 80 90 100`) or use the keyword `fromtoby` (or `fromToBy` or `FromToBy`) to indicate a sequence (e.g. `--T1_vcf_file filename fromtoby 10 100 10`), where the first value is the start of the sequence, (from), the second value is the end of the sequence (to) and the third value is the increment (by). One can also mix up these methods. For example `--T1_vcf_file filename 1 10 20 30 40 50 60 70 80 90 100 107 200 300 400 500 550 600 650 700 750 800 850 900 950 1000` can be rewritten `--T1_vcf_file filename 1 fromtoby 10 100 10 107 fromtoby 200 500 100 fromtoby 550 1000 50`

14.2 Export population to a binary file

It may be of interest to export the population in a binary file. The main reason why you would want to do that would be to reuse the saved population as starting population for another simulation (with the option `--readPopFromBinary`). It can also be useful, to recalculate statistics about this population later via SimBit by simulating 0 generation or (for advanced users) by directly treating the data yourself from a format that takes very little storage.

To export the population in a binary file use the option `--SaveBinary_file`

`--SaveBinary_file file generations` Save Binary files

such as for example

```
--S HomoSapiens PanTroglodytes --SavePopBinary MyBin 1 10 25 FromToBy
 50 500 50
```

will save the files `HomoSapiens_MyBin_G50` and `PanTroglodytes_MyBin_G50`, `HomoSapiens_MyBin_G100` and `PanTroglodytes_MyBin_G100`, `HomoSapiens_MyBin_G500` and `PanTroglodytes_MyBin_G500` at the generations 50, 100 and 500, respectively. At each generation it will also save a binary file with the seed information. Its name is just like the above except the the species name is replaced by 'seed'. This also mean by the way that the 'seed' cannot be used as a species name (SimBit will send an error message if you try).

14.3 User friendly outputs

Outputs that are specific to certain type of loci have it clearly indicated in their name (e.g. `--T1_FST_file` or `--T3_MeanVar_file`). Just like before the outputs start with the name of the file (or the keywords `nfn` or `NFN`) and is followed by the time at which the output is expected. Some options can also take a `subset` keyword that will allow the user to indicate the set of loci over which the output should be computed. For example:
`--T1_FST_file evenLoci 50 100 subset 0 2 4 6 8 10 12 14`
It will outputs data at generations 50 and 100 for loci 0 2 4 6 8 10 12 and 14. For these outputs that can take a `subset` keyword, you can give the option several times to ask for different subset. For example

```
--T1_FST_file evenLoci 50 100 subset 0 2 4 6 8 10 12 14 --T1_FST_file
 oddLoci 50 100 subset 1 3 5 7 9 11 13 15
```

When using the same option several, it is the user's responsibility to give different names to these files otherwise the data will be confounded in the same file in ways that can be confusing. The option `--T1_fitness_file` does not accept the `subset` keyword because a more advanced subsetting solution exists already via the option `fitnessSubsetLoci_file`.

14.3.1 Outputs for T1 loci

`--T1_LargeOutput_file file generations` outputs: T1 Large

It outputs the complete genotype of every individual in a TSV (Tab separated Values). This can be a very large file (hence the silly name). The extension `.T1LO` is added to the filename. For example, `--T1_LargeOutput_file LO 100 200 300 400 500` will print the large outputs for generations 100, 200, 300, 400 and 500.

`--T1_vcf_file filename generations` outputs: T1 VCF

It outputs a VCF (Variant Call Format) file. This can be very handy for usage with the command line `vcftools`. With the help of the software PGDspider, one can reach almost any commonly used file format from this VCF file. Some software using VCF files do not accept that locus or chromosome to have an index of 0. Hence, unlike everywhere else in SimBit, the first locus has index 1 and the first chromosome has index 1. This leads to a shift of 1 when comparing outputs of, say .T1LO files with .vcf files The extension `.T1vcf` is added to the filename. This option accepts the `subset` keyword.

`--T1_AlleleFreq_file filename generations` outputs: T1 Allele Frequencies

It outputs the allele frequency at each locus for each patch. The extension `.AlleleFreq` is added to the filename. This option accepts the `subset` keyword.

`--T1_FST_file filename generations` outputs: F_{ST}

It outputs F_{ST} measures (Weir and Cockerham, as well as Nei estimates for both averaging over all loci and as the ratio of the averages of numerator and denominator over all loci). The extension `.T1FST` is added to the filename. More info can be given via `--T1_FST_info`. This option accepts the `subset` keyword.

`--T1_FST_info [number of patches to consider]`

outputs: F_{ST} info

Gives info about what patch comparisons must be performed for F_{ST} calculations. It is easier to explain with examples. If you input `--T1_FST_info allInteractions 2`, then SimBit will output all pairwise F_{ST} measures. If you input `--T1_FST_info allInteractions 3`, then SimBit will output F_{ST} measures for all possible triplets of patches.

`--T1_MeanLD_file filename generations`

outputs: T1 Mean Linkage Disequilibrium

It outputs the within and among chromosomes average linkage disequilibrium per patch. The extension `.MeanLD` is added to the filename. This option accepts the `subset` keyword.

`--T1_LongestRun_file filename generations`

outputs: T1 Longest Run

It outputs the longest run (=consecutive series) of 0 or longest run of 1 for each haplotype (of each individual in each patch). The extension `.LR` is added to the filename. This option accepts the `subset` keyword.

`--T1_HybridIndex_file filename generations`

outputs: T1 Hybrid Index

It outputs the hybrid index of each individual. Here, I call the hybrid index of an individual, the fraction of T1 loci of this individual that carry the `1` allele. This is a helpful statistic when used alongside `--T1_ini (--T1_Initial_AlleleFreqs )`. It allows the user to specify specific patches where all individuals are fixed for the `0` allele and other patches fix for the `1` allele and see how they interbreed through time. With selection against heterozygotes, you can have some barrier to gene flow. This option accepts the `subset` keyword.

`--T1_ExpectiMinRec_file filename generations`

T1 Average minimal number recombination events

`--T1_ExpectiMinRec` outputs the average (averaged among all haplotypes within each patch) number of times a run of zero is stopped by a run of 1 and vice versa.

For example the haplotype '1111011111100000000' has 3 such events. The extension `.EMR` is added to the filename. This option accepts the `subset` keyword.

14.3.2 Outputs for T2 loci

outputs: T2 Large

`--T2_LargeOutput_file filename ints`

It outputs all genotypes of all individuals. This can be a very large file. The extension `.T2LO` is added to the filename.

14.3.3 Outputs for T3 loci

outputs: T3 Large

`--T3_LargeOutput_file filanem ints`

It outputs all genotypes of all individuals (but not the phenotypes). This can be a very large file. The extension `.T3LO` is added to the filename.

outputs: T3 MeanVar

`--T3_MeanVar_file filename ints`

It outputs the mean and variance in phenotype (along each dimension of the phenotypic space) per patch.

14.3.4 Outputs for T4 loci

For T4 loci, there are two types of outputs. The following two outputs (`--T4_LargeOutput_file` and `--T4_vcf_file (--T4_VCF_file)`) work exactly as the T1 outputs of the same style except that they can't take the `subset` keyword.

outputs: T4 Large Outputs

`--T4_LargeOutput_file filename ints`

outputs: T4 VCF

`--T4_vcf_file (--T4_VCF_file)filename ints`

The outputs `--T4_printTree` t outputs the entire Ancestral Recombination Coalescence Tree for the T4 loci. Note that the tree is being outputted each time the current states are being computed, which is each time you asked for some T4 outputs and each time the average number of nodes per haplotype overpass the limit set by the option `--T4_maxAverageNbNodesPerHaplotype` . Interpreting several trees might be tricky. Hence, if you are not asking for any specific T4 output before the last generation, you might want to set `--T4_maxAverageNbNodesPerHaplotype` to a very large number (like `--T4_maxAverageNbNodesPerHaplotype 1e9` for example) to make sure the current states will never be computed before the very end of the simulation. That might affect performance though if the recombination rate is relatively large.

outputs: T4 Tree

`--T4_printTree filename`

14.3.5 Other Outputs

Outputs: fitness

`--fitness_file filename ints`

it outputs the fitness of every individual in the population. The extension `.fit` is added to the file name.

`--fitnessSubsetLoci_file filename ints @S0 LociSet T1 ints T2 ints T3 ints T1epistasis ints LociSet ints ... @S1 LociSet T2 ints ...`

Outputs: fitness subset genome

This option is very similar to `--fitness_file` except that it allows to output fitness for specific subset of the genome. The option is hence species specific and allows to define an infinite number of subset of genome (called `LociSet`) a user may want. The argument comes like other outputs argument with the filename followed by the time at which outputs must be produced. What follows the time indication is a little bit unusual. First you have the species specific markers. Then, you can create an indefinite number of sets of loci from which fitness will be computed. Each set starts with the keyword 'LociSet'. After this keyword, you specify what types of loci you are willing to consider and their associated indices. The four possible types are T1 T2 T3 and T1epistasis. You can specify several type per set if you want. For example

`--fitnessSubsetLoci_file myFile LociSet T1epistasis 0 3 4 T1 0 5 10`

```
T3 0 1 2 3 LociSet T1epistasis 0 1 2 3 4
```

will lead SimBit to consider two sets of loci. One for which it will compute normal selection on the T1 0 5 and 10 as well as epistatic selection on T1 loci 0 3 and 4 and selection on T3 loci 0, 1, 2 and 3. The second set only computes the epistatic selection on loci 0, 1, 2, 3 and 4. Note that for both LociSet, the 4 components of fitness (T1Fitness, T2Fitness, T3Fitness and T1epistasisFitness) are printed. Hence, it would serves no purpose to add a third LociSet `LociSet T1 0 5 10` and this information is already contained in the first LociSet. The example lack any species specific marker and therefore assumes either a single species or that the same LociSet are required for all species.

Outputs: fitness Stats

```
--fitnessStats_file filename ints
```

it outputs the fitness mean and variance per patch. The extension `.fitStats` is added to the filename

Outputs: patch sizes

```
--patchSize_file filename ints
```

it outputs the number of individuals in each patch. The extension `.patchSize` is added to the filename.

Outputs: Extinction

```
--exctinction_file filename ints
```

it outputs the extinction time (if it applies) for every species.

Outputs: Genealogy

```
--genealogy_file filename ints
```

it outputs the entire genealogy between the two time points indicated (expects only two time points). Because, this represents a lot of data, SimBit does not keep the entire genealogy in the RAM but print it out in file that it later merge together into a single file and then delete. Because a lot of files are being printed during the simulation, we recommend that you indicate a directory to SimBit, Something like

```
--genealogy_file genealogy/familyTree 1000 5000
```

At the end of the simulation, there is a single file left. Each generation is a line that looks like this

```
G_1005 P0I0_P0I34_P3I102 P0I1_P0I121_P0I97 etc...
```

`G_1005` indicates the generation (generation 1005) and is followed by tokens with three values separated by `_` such as `P0I0_P0I34_P3I102`. The first one is and ID for the offspring and the last two are IDs for the two parents. `P0I0_P0I34_P3I102` means that the individual index `0` of patch index `0` is the descendent of a parent from patch index `0` individual index `34` and from a parent from patch index `3` (a migrant) individual index `102`. I would not quite call it a user-friendly output but this format can actually become quite handy esp. that it allows to match individuals as identified here with their other attributes (such as their fitness) as given by other output files.

The option `--coalesce` allows SimBit to directly compute the coalescent tree from the genealogy files. In other words, it removes all individuals that did not leave offspring at the last generation sampled for the genealogy.

`-coalesce int`

Outputs: Coalescence

It takes a single value which is either 0 (which means don't coalesce) and a positive number. If a positive number is used, then SimBit will remove from the genealogy, all ancestors that did not leave any offspring at the last generation sampled. Note that when cloning / selfing rates are high, coalescence happen fast but in presence of sexual reproduction, very few ancestors leave absolutely nothing to the current generation and the option becomes almost pointless.
The positive value chosen matters only for performance reasons. A value of 250 for example, means that SimBit will look back at previous generations (to remove all ancestors that did not leave any offspring) every 250 generations. Because looking back at ancestors is a little bit slow (because the information is kept on the hard drive, not on the RAM), SimBit will run faster if you input a large number. However, keeping a very large geneaology on the hard drive may become problematic and may saturate the hard drive. As such, it may be of interest to remove all ancestors regularly enough so as to free up the storage. A priori, we would suggest that hard drive storage is rarely a limitation, and we would invite our user to use a large enough number. Without having done much

testing, I would a priori recommend using a value of about 4N (where N is the total population size) generations.