SimBit

User Manual

Author:

Remi Matthey-Doret

Important contributor:

Michael C. Whitlock

Main beta tester:

Pirmin Nietlisbach

Last revised December 13, 2019, for version 4.9.11

Contents

1 Correspondance 3

2 How to cite 3

3 A few a priori information 3

3.1 SimBit in a few words 3
3.2 How to obtain SimBit 3
3.3 How to compile SimBit 4
3.4 How to read this manual 4
3.5 How to give arguments to SimBit and Presentation of options in the manual 4
3.6 Performance comparison 5
3.7 Genetic architecture - The basic types of locus 5
3.8 unif and A 7
3.9 seq, seqInt, rep, repeach and fromToBy 8
3.10 Listing all options 9
3.11 SimBit Version 9

4 Species, Generation and Habitat specific options 9

4.1 Species specific options 9
4.2 Generation specific options 11
4.3 Habitat specific options 12

5 Launching basic simulations 13

6 Selection and phenotype 18

6.1 General concepts 18
6.2 T1 19
6.3 T2 21
6.4 T3 22
6.5 T4 23
6.6 T5 23

7 Demography and species ecology 24

8 Mating system 27

9 Defining individual types 29

10 Initial Population 30

11 Reset genetics 32

12 Technical options **34**

13 Performance options **36**

14 Outputs **38**

14.1 Logfile 39

14.2 Export population to a binary file 40

14.3 User friendly outputs 41

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 entire 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 How to cite

The software is not published yet, so just cite the github page *https://github.com/RemiMattheyDoret/SimBit*.

3 A few a priori information

3.1 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 can simulate several species with their ecological interactions too. SimBit has been created with two main ideas in mind: Having a simple user interface with very good error report and to be extremely fast for a wide variety of scenarios. One way SimBit achieve such high performance is by allowing a diversity of internal representation of the genetics of individuals.

3.2 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 .

3.3 How to compile SimBit

Using the command line (terminal), `cd` to the downloaded directory 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) but SimBit can also compile on C++11 if you do "make c++11". SimBit also uses a few boost files that you might need to download if you have not yet.
If you want to be able to call SimBit without having to specify the whole path, just do it manually. In MacOS, you can do `touch ~/.bash_profile; open ~/.bash_profile` and write `export PATH="/PathToSimBit/SimBit/bin/:$PATH"` by changing `/PathToSimBit/SimBit/bin/` with the correct path on your machine.

3.4 How to read this manual

The manual is meant to be read from start to finish. Just take two hours to read it and unleash SimBit's full potential!

3.5 How to give arguments to SimBit and Presentation of options in the manual

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)`.

You do not need to bother about the ordering of the options. 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 message.
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`. If you want SimBit to read all the lines in the file then write 'all' (or just 'a') instead of the line number. For example,
./SimBit file ArgFile.txt a
The advantage of this technic is that SimBit will then ignore anything that follows the `#` sign on a given line allowing the user to leave comment in the input file. See the folder "exampleCommands" as examples.

3.6 Performance comparison

Figures to come...

3.7 Genetic architecture - The basic types of locus

The representation of the genetic architecture is a key factor affecting flexibility and performance. Indeed different types of simulation require different representation of the genetic architecture in order to max out the performance. SimBit offers 5 types of locus that are referred to as T1, T2, T3, T4 and T5, which will be defined below. Loci of different types are integrated on the same recombination map (see the options `--L (--Loci)` and `--r (--recombinationRate)` below). T1 and T5 loci are meant to perform the same types of simulations but with different performance. T1 are meant to be used when there is high per locus genetic diversity while T5 are meant to be used when there is moderate to low genetic diversity per locus.

3.7.1 T1 loci

T1 loci track binary variables (e.g. mutated vs wildtype). SimBit has in memory for each haplotype an array of bits of the length of the number of T1 loci simulated. The nth bit indicates whether the nth T1 locus of this haplotype is mutated or not. T1 loci have high performance for simulations with very high per locus genetic diversity.

3.7.2 T2 loci

T2 loci are meant to represent aggregate blocks of loci and counts the number of mutations happening in this block. This type should be used only when 1) the genetic diversity per T2 locus is very high, 2) when performance is a major concern, 3) you are satisfied with the limited selection scenario it can model and 4) a simple count of the number of mutation happening per T2 locus for each haplotype is a sufficient output for your needs.

3.7.3 T3 loci

T3 loci are quantitative trait loci (QTL) and code for a n-dimensional phenotype. The user can set the phenotypic effect of each T3 locus on each of the n axes of the phenotype (essentially specifying a G-matrix), and this can also be set to be dependent on the environment in order to simulate a plastic response. A user can also add random developmental noise (drawn from a gaussian distribution) in the production of a phenotype in order to reduce heritability. For T3 loci, the user can define a fitness landscape and an individual's fitness is given by its phenotype.

As for the moment a T3 locus is coded as a single byte. It can therefore only take $2^8 = 256$ different values (from -128 to 127). This is therefore only an approximation to a truly perfectly quantitative locus.

3.7.4 T4 loci

For T4 loci, SimBit computes the coalescent tree of the population over time and add the mutations onto the tree when the user asks for output. T4 loci are extremely fast when the recombination rate is low. T4 loci are inspired from ref (already implemented in SLiM; ref). T4 loci are necessarily neutral.

The only advantage of T4 loci over T1 or T5 loci comes from 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 and T5 loci for cases of high recombination and cases with few loci. Note that variance in recombination rate throughout the genome typically helps making T4 faster. As a general advice: don't assume that one is faster but try it out.

3.7.5 T5 loci

T5 loci are very similar to T1 loci. Two simulations with the same seed differing only by the fact that one uses T1 loci and the other uses T5 loci will produce the same output. The big difference is how SimBit tracks their values. For each haplotype, SimBit has an array with the position of each T5 locus that are mutated. T5 loci tend to perform better than T1 loci for moderate to low genetic diversity.
Behind the scene, SimBit will track separately T5 loci that are under selection (which it calls T5sel) and T5 loci that are neutral (which it calls T5ntrl) for improved performance. SimBit can also compress T5 loci (either the neutral ones and/or the selected ones) information in memory. Behind the scenes again, the compressed T5 loci are actually called T6 (T6sel and T6ntrl for selected and neutral loci, respectively) but I don't think you really needed to know that! Compression reduces the RAM usage by (up to) a factor of 2. It can also increase or reduce CPU time depending on the simulation scenario. By default, SimBit makes this compression (on the neutral T5 loci only) only when it is certain it will improve performance (which is when the number of T5 neutral loci is between 10 and 2^{16}). For advanced users, it is also possible to ask SimBit to inverse the meaning of some loci depending on their frequencies. For example, if the locus 23 is fixed or quasi-fixed, then SimBit can inverse the meaning of having the number 23 in its haplotype description. As a result a haplotype would track this 23rd locus only if they carry the non-mutated allele. These advanced performance tweaks are explained in the section "performance options".

3.8 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" saying that you want to input as many values as needed. "unif" stands for "uniform" saying that you want all elements to be set to the same value. As an example, to set the carrying capacity for four patches (`--PN 4` or `--PatchNumber 4`) to 1000, then you could do `--N unif 1e3` or equivalently `--N A 1e3 1e3 1e3 1e3`.

3.9 seq, seqInt, rep, repeach and fromToBy

There are special keywords `seq`, `seqInt`, `rep`, `repeach` and `fromToBy`. 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`, `seqInt`, `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.
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`. 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.

The keywords can mixed at will. For example `3 4 rep hello 3 0 seq 1 2 0.3` is equivalent to `3 4 hello hello hello 0 1.0 1.3 1.6 1.9 3`. 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.

The keyword `fromToBy` (or `fromtoby` or `FromToBy`) can only be used for output related option (see section `Outputs`). `fromToBy` works exaclty like `seqInt` excepts that it accepts the keyword `end` to indicate the last generation of the simulation. For example `--fitnessStats_file fitFile fromToBy 0 end 10` asks for the output file showing the fitness summary statistics, the file name will be "fitFile" and the outputs will be printed every ten generations from generation 0 and up to the last generation of the simulation.

3.10 Listing all options

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

3.11 SimBit Version

Everytime you run the executable, SimBit prints the version (bottom right of the logo) in standard output.

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`. If you want to simulate a single species, a single type of habitat (no environmental heterogeneiity) and no change over time, then you do not have to bother but these `@G`, `@H` and `@H`.
All these species, habitat and generation specific markers must come in order. For example, `--N @G0 50 @G130 1000` is correct but `--N @G130 1000 @G0 50` leads to an error message. Similarly, habitats that are named by indices must come in order. For example, `--T1_fit @H0 unif 1 1 0.99 @H1 unif 1 1 0.9` is correct but `--T1_fit @H0 unif 1 1 0.99 @H1 unif 1 1 0.9` leads to an error message. Finally, species are named but you have to follow the ordering you used when naming these species. For example `--S Quercus Fagus --N @SQuercus A 1000 @SFagus A 60000` is correct but `--S Quercus Fagus --N @SFagus A 60000 @SQuercus A 1000` leads to an error message.
If you need more information on these markers, you can read the following three sections ("Species specific options", "Generation specific options" and "Habitat specific option"). Otherwise, you can just skip them and read "Launching basic simulations".

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 architecture, selection scenarios and demography are species specific.
You need to name your different species with the option –S (–species).

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

Species

Each species needs a unique name. If you do not use the option `--S` SimBit, will assume you want a single species.
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
```

and

```
--PatchNumber 1 --S Quercus Fagus --N @SFagus A 1000 @SQuercus A 60000
```

are NOT 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 A 1000
```

If you do not specify a given species then SimBit assumes you want to same thing than you asked for the previous species. As such,

```
--PatchNumber 1 --S Quercus Fagus --N @S0 A 1000
```

is also valid. 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

For generation specific options, you can use the @G marker directly followed (without space) by the generation. Let's consider 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

```
--nbGenerations 2000 --PatchNumber @G0 1 @G1000 5 @G1200 1
```

or

```
--nbGenerations 2000 --PatchNumber 1 @G1000 5 @G1200 1
```

but you cannot write

```
--nbGenerations 2000 --PatchNumber @G1000 5 @G0 1 @G1200 1
```

If you wish to have 5 patches the whole simulation the input would be

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

or simply

```
--PatchNumber 5
```

Here is another example

```
--nbGenerations 2000 --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 throw an error message and abort. For example the following would return an error

```
--nbGenerations 2000 --PatchNumber @G0 1 @G1200 3 --N @G0 unif 100
@G500 A 200 150 200
```

The reason it fails is the from generation 500 to generation 1000, a single patch size is indicated but from generation 500 to 2000, there are patch capacity indicated for three patches. This causes a mismatch between generations 500 and 1200.

4.3 Habitat specific options

All options that relate to the selection scenario (such as `--T1_fit` for example) and phenotypes (`--T3_pheno`) are habitat-specific. A habitat has to be understood in its ecological definition. A "habitat" could also have been called an "environment". 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 --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. 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 (`--PN (--PatchNumber)`), the number of individuals per patch (`--N (--PatchCapacity)`), the number of loci, their types and physical ordering on the chromosome (`--L (--Loci)`), the number of generations (`--nbGens (--nbGenerations)`), the mutation rate for the type of locus indicade to option `--L (--Loci)` (`--T1_mu (--T1_MutationRate )`, `--T2_mu (--T2_MutationRate)`, `--T3_mu (--T3_MutationRate)`, `--T4_mu (--T4_MutationRate)`, `--T5_mu (--T5_MutationRate)`), and the dispersal rate (`--m` or `--DispMat`; only required if there is more than one patch). By default the recombination rate will be set at zero (perfect linkage). Note by the way 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 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

```
--PN (--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 Patch-

Number for this specific range of generations. Note that by default, the patch size is always at its carrying capacity.

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 T3 loci, followed by 1000 T1 loci followed by 12 T3 loci, you could input

```
--L T3 23 T1 1e3 T3 12
```

Note that instead of `T3 12`, you could have `T3 6 T3 4 T3 2` and it would be equivalent. Lower case T are also accepted. You can also ignore the Ts (e.g. `--L 3 23 1 1e3 3 12` but it not very human readable).
The recombination rate between any of these loci, including, the placing of loci on independent chromosomes is indicated via the option `--r (--RecombinationRate )`. Because there are above 23 + 1e3 + 12 = 1045 loci, the option `--r` will expect 1044 entries (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

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'	Yes
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.	No
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

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, --T4_mu and --T5_mu are not present in the above code but I will mention it here for their similitude with --T1_mu.

```
--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

```
--T5_mu (--T5_MutationRate)@S0 Mode float ...
```

Mutation rate on T5 loci

The last option in the above example is --r. This options specifies the recombination rate between any two loci (whatever their type). 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. The number of entries should equal the total number of loci minus 1. For example --r rate unif 1e-7 sets the recombination rate between any two adjacent loci to 1e-6. To indicate perfectly indepedent loci (like a chromosome break), just give it a rate of 0.5. If you are using M or cM, you can just say -1 and it will be understood as a chromosome break.

6 Selection and phenotype

6.1 General concepts

The selection scenario can be set independently for each type of locus with options `--T1_fit`, `T1_epistasis`, `--T2_fit`, `--T3_fit` and `--T5_fit` (T4 loci are always neutral, there is therefore no `--T4_fit`). Fitness effects among locus types are multiplicative.

For a number of types of loci (see below), SimBit can make use of an assumption about the selection scenario that can provide substantial improvement in run time. I call this assumption the "multiplicative fitness" assumption (abbreviated "multfit")". The multiplicative fitness assumption assumes that the fitnesses of the three possible genotypes are 1, $1 - s$ and $(1 - s)^2$. With this assumption, dominance coefficients are close to 0.5 especially for low selection coefficients. For examples, if the double mutant homozygote fitness is $1 - t = 1 - 0.001$, then $h \approx 0.5001$. If $1 - t = 1 - 0.1$, then $h \approx 0.51$. When taking advantage of the assumption of multiplicative fitness, SimBit partitions a haplotype into blocks and computes the fitness value for each block. If, during reproduction, no recombination events happen within a given block, then SimBit will not need to recompute the fitness for this specific block as the fitness of the block can simply be multiplied over by the fitness of the same block on the other haplotype. This technique yields substantial performance improvement in terms of CPU time especially when recombination rate within blocks is relatively low. SimBit does a decent job at choosing the size of blocks but a user can have complete control over the block sizes with the option `--FitnessMapInfo` (see section "Performance options"). Unless the exact dominance relationship is of central importance, it is generally recommended to make use of this assumption (especially when recombination rate is low and when there are a fair amount of loci).
All selection scenarios described below (including epistasis) are habitat specific, hence allowing any kind of spatial and temporal variation in selection pressures (as a reminder, the matching between patches and habitats with option `--H` is generation specific). By default, selection happens on fertility, but it can also be

simulated on viability or on both fertility and viability.

```
--selectionOn <info>
```

The info are either `fertility`, `viability` or `both`. Selection on fertility is faster than selection on viability. It is therefore the default and recommended mode.

Selection on fertility or viabilityi

6.2 T1

On T1 loci a user can either set the fitness values of each of the three genotypes, or take advantage of the multiplicative fitness assumption and only provide a single fitness value per locus.

```
--T1_fit (--T1_FitnessEffects)@S0 @H0 Mode float @Hx ..
```

Selection on T1 loci

There are five possible Modes; `A`, `domA`, `cstH`, `unif`, `multfitA` (aka. `MultiplicityA`), `multfitUnif` (aka `MultiplicityUnif`), `multfitGamma` (aka `MultiplicityGamma`). All Modes starting by "multfit" (or "multiplicity" for alternative names) means that you are willing to assume "multiplicative fitness" 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. Table 2 summarizes the different mode of entries.

Note that in the current version, it is impossible to make the "multfit" assumption for a given habitat but not for another one. It will raise an error message. There is no good reason for this limitation. If you need to get rid of it, you can ask me for help.

For epistatic interactions use

```
--T1_epistasis (--T1_EpistaticFitnessEffects)@S0 @H0 loci <list loci
> fit <list fitnesses> loci <list loci> fit <list fitnesses> etc...
```

Epistasis

```
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

Table 2: Input of data for –T1_fit

<table>
<tr><th>Mode</th><th>Meaning</th><th>Example</th><th>Number of entries</th></tr>
<tr><td>A</td><td>Indicates the fitness of all three genotypes at all loci</td><td><code>A Locus1_G0|0 Locus1_G0|1 L1G1 |1 Locus2_G0|0 Locus2_G0|1 Locus2_G1|1</code></td><td>3 * number of T1 loci</td></tr>
<tr><td>cstH</td><td>first indicate a dominance coefficient 'h', then indicate the keyword 'hetero' or 'homo'. Then indicate for each locus the fitness of either the heterozygote or the double mutant homozygote fitness (depending on the keyword). The fitness of the other genotype will be computed from h. The fitness of the double wild type is always 1.0</td><td>e.g. <code>0.5 hetero Locus1_Ghetero Locus2_Ghetero Locus3_Ghetero</code></td><td>1 + 1 + number of T1 loci</td></tr>
<tr><td>domA</td><td>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.</td><td><code>domA fun 0.2 Locus1_G011 Locus2_G1|1 Locus3_G1|1</code></td><td>number of T1 loci + cst or fun + $\bar{h}$</td></tr>
<tr><td>unif</td><td>Indicates the fitness components for all three genotype and assume that all loci are under the same selection scenario</td><td><code>unif G0|0 G0|1 G1|1</code></td><td>3</td></tr>
<tr><td>multfitA</td><td>Indicates the fitness components of the genotype 0|1 at each locus separately and makes the assumption of multiplicative fitness</td><td><code>MultiplicityA Locus1 Locus2 Locus3</code></td><td>Number of T1 loci</td></tr>
<tr><td>multfitUnif</td><td>Indicates the fitness components of the genotype 0|1 for all loci makes the assumption of multiplicative fitness</td><td><code>multfitUnif G0|1</code></td><td>1</td></tr>
<tr><td>multfitGamma</td><td>Just like <code>multfitA</code> 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</td><td><code>multfitGamma alpha beta</code></td><td>2</td></tr>
</table>

Table 3: Ordering of fitness values for and three-locus epistatic interactions

locus A genotype:	00	00	00	00	00	00	00	00	00	01	01	01	01	01	01	01	01	01	11	11	11	11	11	11	11	11	11
locus B genotype:	00	00	00	01	01	01	11	11	11	00	00	00	01	01	01	11	11	11	00	00	00	01	01	01	11	11	11
locus C genotype:	00	01	11	00	01	11	00	01	11	00	01	11	00	01	11	00	01	11	00	01	11	00	01	11	00	01	11

such thing or not. If this is the case SimBit will throw a warning message though just to make sure you know what you are doing. Note also that one locus can belong to more than one set of loci for which an epistatic interaction is defined.

The user can specify any number of set of loci that are in epistatic interactions and each set can contain any number of loci. If you specify n loci after keyword `loci`, then SimBit will expect 3^n fitness values after the keyword `fit`. Here is a table to explain the ordering of the fitnesses to input for a three-locus interaction. For a three-locus interaction , SimBit expects $3^3 = 27$ fitness values. The first fitness value entered is the fitness for when the individual is homozygous wild type for the three loci. The
For example,

```
--T1_epistasis loci 0 5 fit 1 0.9 0.8 0.9 0.9 0.9 0.8 0.9 1 loci 2
3 1 0.9 0.8 0.8 0.9 1 1 0.9 0.8
```

In this example, the loci 0 and 5 have an additive by additive epistatic interaction while the loci 2 and 3 have an additive by dominance epistatic interaction.

6.3 T2

For T2 loci the logic is very similar

```
--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 multfit of dominance effects. (One may argue that the Modes might better be renamed multfitA, multfitUnif and mulfitGamma).

6.4 T3

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 T3 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 phe-

notype 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 ...`).

6.5 T4

T4 loci are fundamentally neutral.

6.6 T5

Selection on T5 loci

```
--T5_fit (--T5_FitnessLandscape)@S0 @H0 Mode float @Hx ..
```

The modes for selection on T5 loci are very similar than on T1 loci. There are five possible Modes; `A`, `domA`, `cstH`, `unif`, `multfitA` (aka. `MultiplicityA`), `multfitUnif` (aka `MultiplicityUnif`), `multfitGamma` (aka `MultiplicityGamma`). The only difference with T1 loci is that if you do not want to take advantage of the multfit assumption, then you can specify only two fitness effects per locus, the fitness effect of the heterozygote individual and of the double mutant homozygote (in this order). The fitness of the homozygote wild type is always assumed to be 1.0 for T5 loci.

7 Demography and species ecology

Here I do not mean to talk much about the basic options `--N` and `--m` (see section "Launching basic simulations" for their uses). Here, I want to talk about how change in patch size is modelled and how fecundity and selection, species interaction, and migration affect that.

SimBit assumes non-overlapping generations (although different species can have different generation times) and assumes discrete patches (although patches can be made arbitrarily small, essentially mimicking continuous space). Outside of these two assumptions, SimBit can simulate very diverse types of demographies. SimBit can simulate any number of patches with any migration matrix (see option `--m` in section "Launching basic simulations"), carrying capacity (see option `--N` in section "Launching basic simulations"), variation of the patch size from the carrying capacity based on realized fecundity with exponential or logistic growth model (the growth model can be set for each patch independently; see more on that below). Each patch can be initialized at the desired size and all of the above parameters can vary over time.

Dispersal can happen at the gametic or at the zygotic phase and may be a function of the patch mean fitness (hard vs soft selection). This is modified with option `--gameteDispersal`.

Selection softness

```
--DispWeightByFitness 'bool'
```

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 (see below), then dispersal is necessarily weighted by fitness. Two entries are possible `0` (or `false` or other equivalent) and `1` (or `true` or other equivalent). `false` is default and means soft selection and `true` means hard selection. Choosing `0` might make the simulation a tiny bit faster especially for simulations with a lot of generations.

`--gameteDispersal @S0 bool ...`

gamete dispersal

If the option 'gameteDispersal' is set to 'true', 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 offspring 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'.

fecundity

`--fec (--fecundityForFitnessOfOne)@S0 float ...`

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 (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 hermaphrodites a fecundity of at least 1 will be necessary to replenish the entire patch of individuals (there can be stochastic variation that will cause on average a decrease in the patch size; see `--stochasticGrowth`). If you use a males-and-females mating system with a sex ratio of say, 0.75 (3 male for 1 female; see `--matingSystem`), then you will need a fecundity of at least 4 to have subsistance.

SimBit can simulate realistic change in population in response to patch mean fitnesses. Let us denote at time t the expected number of offspring of a species s produced in patch p as $P_{t,s,p}$. Let us also denote the growth rate $r_{t,s,p} = f \sum w_i$ that is the product of the theoretical maximum fecundity of an individual having a fitness of 1.0 (f; set by the user) and the sum of finesses in this patch ($\sum w_i$). If the user allows the patch size to vary from the carrying capacity of this species at that time $K_{t,s,p}$ then the expected number of offspring produced patch size at the next generation is set as $\bar{P}_{t,s,p} = rN_{t,s,p}$ for the exponential model and $\bar{P}_{t,s,p} = N_{t,s,p} + rN_{t,s,p}\left(1 - \frac{N_{t,s,p}}{K_{t,s,p}}\right)$ for the logistic model, where $N_{t,s,p}$ is the size of the patch p of species s at time t. The actual number of offspring produced, $P_{t,s,p}$ can then either be set deterministically $\left(P_{t,s,p} = \bar{P}_{t,s,p}\right)$ or stochastically

$\left(P_{t,s,p} = Poisson\left(\bar{P}_{t,s,p}\right)\right)$. With more than one patch, these offspring produced are then spread out through migration. With a single patch (or in absence of immigration and emigration for the patch p), $N_{t+1,s,p}$ is simply set to $P_{t,s,p}$. Note that I here use $N_{t,s,p}$ to denote the patch size and not the carrying capacity which is set with option `--N` (which is a little confusing, sorry).

Into the above framework, we can add the fact that different species can affect each others patch size through their ecological relationships. This can be achieved through a competition matrix and/or through an interaction matrix. Let $\alpha_{i,s}$ be an element of the competition matrix describing the competitive effect of species i on focal species s. The expected number of offspring produced is then given by $\bar{P}_{t,s,p} = N_{t,s,p} + rN_{t,s,p}\left(1 - \frac{\sum_i \alpha_{i,s}N_{t,i,p}}{K_{t,s,p}}\right)$. Note that competitive effects can only be set on species and on patches having logistic growth. Let now $\beta_{i,s}$ be an element of the interaction matrix describing the effect of species i on species s. The interaction effect is added to the expected number of offspring produced $\bar{P}'_{t,s,p} = P_{t,s,p} + \sum_i \beta_{i,s}$. In this equation, I assumed that all effects $\beta_{i,s}$ are independent of the patch sizes of both the causal and recipient species but in practice a user can specify for each $\beta_{i,s}$ whether the effect should be multiplied by the causal species patch size ($N_{t,i,p}$), by the recipient species patch size ($N_{t,s,p}$) or by both. SimBit enforces that all the diagonal values $\alpha_{s,s} = 1.0$ and that all the diagonal values $\beta_{s,s} = 0.0$.

The growth model can be independently for each patch set with the option

Pop growth model

```
--popGrowthModel @S0 mode <value(s)> @Sx ...
```

The two possible modes are 'unif' and 'A'. The value(s) accepted are either the keyword "logistic" (aka. -2), the keyword "exponential" (aka. -1) or any positive integer value. A positive integer value means that you want a logistic growth but you want to set the carrying capacity for this growth calculation to some other value than the carrying capacity set by option `--N`. This is a neat way to allow more realistic demographies such as overshooting of the carrying capacity for example. Note that the patch size can never be greater than what is set with option `--N`. By default the growth rate is logistic. This growth rate only matters if the fecundity (set in option `--fec`) differs form -1.

Stochastic growth

```
--stochasticGrowth @S0 bool @S1 ...
```

If set to `true` (or other equivalent such as `1`, `t`, `T`, ...), then the number of individuals in the next patch is drawn from a Poisson distribution as explained above. By default, it is set to `false`.

Species ecology

```
--eco (--speciesEcologicalRelationships)interaction <interaction matrix
> competition <competition matrix>
```

The ordering of the elements of the interaction and competition matrices respectively makes sense by considering the ordering of the species as entered in option `--S (--Species)`.
For the competition matrix SimBit expects S^2 $\alpha_{i,j}$, where S is the number of species. For the effect of a species on itself (that is all the diagonal $\alpha_{s,s}$), SimBit expects the keyword `self`.
Each element of the interaction matrix is made of two entries; The first entry is the "type" (either `A`, `B`, `C`, `D` or '0') and the second entry is the $\beta_{i,j}$ value. Just like for the competition matrix, for the effect of a species on itself, SimBit expects the keyword `self` (no letter, no $\beta_{i,j}$, just write `self`). The "type" indicates whether the interaction effect must be multiplied by the causal species patch size ($N_{t,i,p}$) (`B`), by the recipient species patch size ($N_{t,s,p}$) (`C`), by both (`D`) or by none (`A`). The type `0` (the number zero, not the letter o) means no interaction (which is the default).
If you want to use the default matrix, just enter `default` as matrix description. The default input is therefore `interaction default competition default`, which would lead to simulate different species that are completely independent.

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

Selfing Rate

```
--selfingRate @S0 float ...
```

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 precedence 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).

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

Generation time

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

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".

9 Defining individual types

As a user, you can define individual types, that is an individual with a perfectly defined genome. These individual types can then be used to initialize the population (see section "Initial Population") or to reset the genetics of the population during run time (see section "Reset genetics").

```
--indIni (--individualInitialization)ind <individualName> haplo0 <haplotype
 description> haplo1 <haplotype description> ind <individualName> ...
```

Individual types

Each individual description starts with the keyword `ind`. It is then followed the name of this individual type by the keyword `haplo0` with the haplotype description and the keyword `haplo1` and its haplotype description. It is also possible to do `ind bothHaplo <haplotype description>` if both haplotypes are to be identical (perfect homozygosity). As an example, the haplotype description is `T1 0 0 0 0 1` indicates that the first 4 loci of T1 loci carry the wildtype allele while the last locus carries the mutated allele. Let's say we want to define an individual type named `wildType` and another named `mutant` that are fixed for the 0 and the 1 allele, respectively. We would do

```
--L T1 50 --indIni ind wildType bothHaplo T1 rep 0 50 ind wildType
mutant bothHaplo T1 rep 1 50
```

Note that there is no mode of entry, so you cannot do `bothHaplo T1 unif 1` but need to use `rep` instead if you do not want to write the number 1 50 times.

Haplotype description must be made for each type of loci that you asked for with option `--L (--Loci)`. For example, you asked for the types T1, T3 and T5 with `--L (--Loci)`, then you can simply do `T1 <T1 loci description> T3 <T3 loci description> T5 <T5 loci description>`. Note that the ordering does not matter (`T5 <T5 loci description> T1 <T1 loci description> T3 <T3 loci description>` is also valid). The following table explains how to provide a description for each type of loci.

Table 4: How to provide haplotype description for each type of locus to option –indIni

Locus type	Entry	Example
T1	Indicate the value of each locus	`T1 0 1 0`
T2	Indicate the number of mutation in each T2 block	`T2 0 0 12 2`
T3	Indicate the value of each QTL	`T3 0 0 5`
T4	Sorry, –indIni is currently not able to set T4 loci.	-
T5	Indicate the position of each mutation	`T5 0 23`

10 Initial Population

By default the patch size is initiated at carrying capacity (set by `--N (--PatchCapacity )`). To set the initial patch size for each patch, use the following option

Initial patch size

```
--InitialpatchSize ints
```

The expected number of values should equal the number of patches at the start of the simulation and no initial patch size should be be larger than the initialicarrying capacity at generation.

By default, all T2 loci are set to carrying 0 mutations and all T1and T5 loci are set to 0. It is possible however to indicate the per patch allele frequency with `--T1_ini (--T1_Initial_AlleleFreqs)` and `--T5_ini (--T5_Initial_AlleleFreqs )`.

T1 Initial allele freqs

```
--T1_ini (--T1_Initial_AlleleFreqs)Mode (value)
```

T5 Initial allele freqs

```
--T5_ini (--T5_Initial_AlleleFreqs)Mode (value)
```

There are four Modes; `AllOnes`, `AllZeros`, `A`, `Shift`. See table for more information. The parenthesis around "value" (`value`) above indicates that the input of values depend on the Mode.

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

Import population

```
--ReadPopFromBinary file
```

Table 5: Input data for –T1_ini and –T5_ini

Mode	Meaning	Example	Default
AllZeros	Set all loci to 0	`AllZeros`	Yes
AllOnes	Set all loci to 1 (this option is not available for T5 as it is a bad idea for performance reasons to have all T5 set to 1.	`AllOnes`	No
A	Specify the per patch and per locus allele frequency. The initialization will be done in a pseudo random manner in order to keep LD low	`A Locus0FreqPatch0 Locus1FreqPatch0 Locus2FreqPatch0 ... Locus0FreqPatch1 Locus1FreqPatch1 Locus2FreqPatch1`	No
Shift	Specify a given patch before which allele frequencies (at all loci) are set to 0 and after which allele frequencies (at all 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. Note that first patch is the patch 0.	No

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. Note that in the current version, T4 loci cannot be dumped into a binary file and therefore can be read from it either.
Finally, one can use individual types (defined with option `--indTypes`; see section "Defining individual types") to initialize a population with option `--indIni (--individualInitialization)`

Import population

```
--indIni (--individualInitialization)patch0 <indTypeName> <nbIndividuals
> <indTypeName> <nbIndividuals> .. patch1 <indTypeName> <nbIndividuals
> ...
```

Here, use the keywords `patch0`, `patch1`, ... `patchx` to describe the initilaization of each patch. In each patch name an individual type and the number of indivdual of this type to put in this patch. SimBit wil ensure that the number of individuals that you put in a patch is equal to the patch size at the beginning pf the simulation (reminder: if the initial patch size is not specified, then it is set to the carrying capacity).

11 Reset genetics

This option deserves its own section. It can either

Reset genetics

```
--resetGenetics @S0 <eventType> <eventDescription> <eventType> <eventDescription
> ... @S1 ...
```

There are two types of events called `eventA` and `eventB`. `eventA` refers to the input of specific mutations. `eventB` refers to the input of individual types (defined with option `--indTypes`).
For `eventA`, SimBit expects an input formatted as `<generation> <TraitType> <typeOfMutationsIfNeeded> <lociListInformation> <haplotypesInformation > <patchAndIndividualInformation> .`

Directly after the keyword `eventA` comes the trait type to be affected in this event (T1, T2, T3 or T5; T4 not accepted). 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 or T5, 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.
For `eventB`, the input is formatted as `<generation> <indTypeName> <nbIndividuals> <indTypeName> <nbIndividuals> ...`. If the patch size differs from the carrying capacity, SimBit will start by adding the individual types in the patch until it reaches carrying capacity, then only will SimBit start to replace currently existing indivdiuals with the individual types.

12 Technical options

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.

A user can terminate a simulation upon some condition depending on the stat of the population simulated.

Kill on demand

`--killOnDemand @S0 <functionToCall> <args> @S1 ...`
There is for the moment, only one function available, it is called `isT1LocusFixedAfterGeneration`. The arguments expected are the T1 locus to consider and the generation after which to call the function. The function kills the simulation if the chosen T1 locus

Table 6: 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

is found fixed anytime after the generation indicated. The option was meant to make it easy for users to implement their own function (by modifying the `void KillOnDemand::readUserInput(InputReader& input)` function in KillOnDemande.cpp and add a new function with the desired name in this same file) but honestly I did not do a great job to give it a nice design.
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

```
--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`.

`--printProgress bool` Print progress

If `true`, SimBit prints on standard output the progress of the simulation (prints the generation, more generations are printed at the beginning of the simulation than later on). By default, it is set to `true`.

13 Performance options

All performance options do not change how the simulation happen. To the exception of `--swapInLifeCycle`, all options will produce the exact same outputs for whatever value of the performance option.

`--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 "multfit". There are two modes `prob` and `descr`. The default entry is some approximation fo `prob 0.008`. SimBit sums up the recombination rate between loci until it reaches the probability indicated before creating a new fitness map block. With `descr`, you can specify exactly which block each locus belong to. It works as `descr <nbLociFirstBlock> <nbLociSecondBlock> <nbLociThirdBlock> ....` The total number of loci must be the total number of loci all types confounded.

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_maxAverageNbNodesPerHaplotype @S1 float @S2 ...
```

T4 loci performance tweak

By default this option is set to 100.

As a reminder, For T5 loci, each haplotype tracks the indices of the loci that have been mutated. When T5 locus, say index 23, reaches fixation, then every haplotype tracks this locus 23, which is not optimal. Instead SimBit can flip the meaning of having 23. If SimBit flips the meaning for index 23, it means that every haplotype that carry the index 23 are not mutated at this locus while those who do not carry index 23 are mutated. SimBit can do this flipping not only for fixed loci but for any desired frequency greater than 0.5. To set this frequency above which meaning of haplotypes are flipped use the following option

```
--T5_freqThreshold (--T5_frequencyThresholdForFlippingMeaning)float
```

T5 loci meaning flip freq

By default, this frequency is 1.0. To set how often SimBit will check for the loci that have reached a high frequency, use `--T5_toggleMutsEveryNGeneration int`

T5 loci meaning flip generation

A value of -1 means "never try to flip meaning". By default, SimBit never tries to flip meaning.

```
--T5_compress (--T5_compressData)bool bool
```

Compress T5

The first bool tells whether the "T5ntrl" must be compressed and the second bool tells whether the "T5sel" must be compressed. By default, "T5sel" are never compressed and "T5ntrl" are compressed if the number of T5 loci is lower than $2^16 - 1 = 65535$.

```
--swapInLifeCycle @S0 bool @S1 ...
```

Avoid copying last reproduction of haplotype

SimBit can either copy each parental haplotypes to create each offspring haplotypes. But if a given parental haplotype creates its last offspring haplotype without recombination, then it would be faster to just copy a pointer to this haplotype. The down side of this is that by copying pointers, we reduce memory contiguity and also we have to figure out when is the last reproduction event of each haplotype. By default, `--swapInLifeCycle` is set to true only if the total number of loci is greater than 100 and if the total recombination map is shorter than 10cM.

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

```
--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

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

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.

outputs: T1 Allele Frequencies

`--T1_AlleleFreq_file filename generations`

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.

outputs: F_{ST}

`--T1_FST_file filename generations`

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). This output file has not really been tested yet. Please consider it with precaution. The extension `.T1FST` is added to the filename. More info can be given via `--T1_FST_info`. This option accepts the `subset` keyword.

outputs: F_{ST} info

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

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.

Outputs: fitness subset genome

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

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.