Manual of CropMetaPop

Contents

In the agricultural world where the major issue is to be able to produce steadily despite climatic, economic and environmental changes, the use of diversified agro-ecosystems (at the inner and intra-varietal level) is a major solution as they offer a better resilience when facing fluctuating environmental conditions. Among the elements that act on diversity dynamics, one can find the circulation of seeds between farmers.

The objective of this project is to create a software allowing to simulate the evolution of a cultivated meta-population thanks to the different evolutionnary forces (drift, selection, mutation, ...). The difference of this tool compared to others already existing is that we include our model in the farmer seed exchange point of view, which is modeled as a migration from a group of seeds of a population to an other.

In a first time, the simulations allowed to evaluate our model by comparing its results to theoretical results. Then, simulations have been done in order to compare the impact of the exchange network on diversity inside the meta-population. These results are preliminary and must be further developped, though they allow to observe significant differences of diversity between the different tested networks.

These first tests being comforting, the software will be used in a more intensive way in order to simulate more complex scenarios which would emphasize the role of certain parameters on cultivated diversity

1 Introduction

1.1 Scope

CropMetaPop is a forward-time, individual-based, genetically explicit stochastic simulation program. This program aims to take into account farmer's practices that shape the evolution of crop genetic diversity by following a metapopulation framework. It accounts for demographic processes like population growth, extinction, colonization, and evolutionary forces like migration, mutation, genetic drift and selection. It allows the simulation of diploid individuals and mixes mating systems from selfing to open-pollinated system.

1.2 Availability

The website and the user's manual of CropMetaPop is available at `https://sourcesup.renater.fr/www/cmp/`.
Its source code can be found at `https://pypi.org/project/cropmetapop/#files`

1.3 Technical

CropMetaPop is based on simuPOP (`http://simupop.sourceforge.net`), a python library for the simulation of populations. CropMetaPop is a console program writen in python3 using an object oriented approach. CropMetaPop can be used on any computer platform. Limits of CropMetaPop are the same as simuPOP. (`http://simupop.sourceforge.net/Main/FAQ#toc4`)

1.4 Licence

1.5 Acknowledgements

The program has been developped within the framework of the EU H2020 project Diversifood under the supervision of Isabelle Goldringer, Mathieu Thomas and Frederic Hospital. We are grateful to the research team DEAP and the group MIRES-MADRES and in particular Jérôme Enjalbert and Francois Massol for their valuable assistance. These people helped us to build and improve CropMetaPop through discussions.

1.6 Authors

This program has been mainly developped by Anne Miramon. Baptiste Rouger contributed mainly to debugging and code readability.

1.7 Main features

• **Meta-population**: A meta-population in CropMetaPop is a set of cultivated populations connected to each other through migration and/or colonization. A population is grown within a patch that corresponds to a farmer's field. A population is composed of individuals from the same species. Pollen flow between populations (fields) is not considered in this version of the model.
• **Seed circulation**: In this program, migration and colonisation of individuals correspond to seed lot circulation among connected patches (fields), i.e. seed lot circulation among fields owned by farmers who belong to the same social network. The rate of migration/colonization corresponds to the probability of seed transfer between two fields that are socially connected. The number of moving seeds is calculated as a function of the parameters of the seed transfert model that checks for seed transfer feasability.
• **Selection**: The selection in CropMetaPop is applied on a quantitative trait. This quantitative trait is defined by a set of gene markers with an additive determinism among markers. The allelic effects at each marker has to be set explicitly and can be different according to populations.

1.8 Input and Output

1.8.1 Input

CropMetaPop is launched using a settings file. The settings file is a text file with flexible and easy to understand structure. The information is specified in a parameter:argument scheme, where the order of the parameters does not matter. The file can be edited with any text editor, and annoted for helping the reader using '#'.

1.8.2 Ouput

CropMetaPop produces the raw results of the simulation. These results summarize genotypes information for every replicate, generation and population like number of individuals for every mono-locus genotype or multi-locus genotype or haplotype. Output also proposes history of every seed lot transfer event (colonization and migration).

2 Using CropMetaPop

This section describes how to install and use CropMetaPop.

2.1 Installation

The source code of CropMetaPop is available on the website `www.cropmetapop.org`. It requires Python3, and the following Python3 librairies:

- **simuPOP** (version $\geq$ 1.1.10): CropMetaPop uses simuPOP[4] to simulate the evolution of populations. (`http://simupop.sourceforge.net`)
- **igraph**: CropMetaPop uses the python module igraph to create default networks. If the network is directly given, this module is not necessary. (`http://igraph.org/python/`)
- **numpy**

A standalone version of CropMetaPop also exist in the website. An up-to-date version can also be found on SourceSup.

2.1.1 For Linux and MacOS

The two libraries required by CropMetaPop can be installed using pip with the following commands in a terminal:

```
$ pip3 install simuPOP
$ pip3 install python-igraph
$ pip3 install numpy
$ pip3 install cropmetapop
```

2.1.2 With Anaconda (especially for Windows)

You first need to install Anaconda (`https://www.anaconda.com/`). We recommend to create a new environment for CropMetaPop using in the Anaconda prompt:

```
$ conda create -n cropmetapop
```

You can then activate and update your environment:

```
$ conda activate cropmetapop
$ conda update conda
```

Next, install the required libraries:

```
$ conda install numpy
$ conda install simupop
$ conda install python-igraph
$ conda install cropmetapop
```

2.2 Launching CropMetaPop

CropMetaPop is a console program. A simulation is defined using a settings file. The settings file is then passed as parameter in the console when CropMetaPop is launched.

```
$ python3 CropMetaPop.py settings.txt
```

2.3 Input Files: settings of simulation

The settings file is a text file with one parameter per line in a 'key:value' scheme.
For example: `generations:5` sets the parameter generation to 5. The order of appearance of the parameters in the settings file does not matter. If a parameter appears several times, only the last one will be taken into account.

2.3.1 Default value

Most of the parameters have default values. The default value of a parameter is taken into account when the parameter is not filled in the settings file. The default values are given in this manual and are specified by '(default:)'. Providing default values help in keeping the settings file short and clear. All these default values are stored in the 'default_setting.txt' file located in the source code.

2.3.2 Parameter types

Parameters may be of different types: integer, double, string, vector or external file. The type of a parameter is specified in brackets '[ ]'.

2.3.2.1 Integer Integer is a whole-number. Two forms are accepted: 1000 or 1e3.

2.3.2.2 Double Double is a floating-point number. Two forms are accepted: 0.001 or 1e-3.

2.3.2.3 String String is a text argument.

2.3.2.4 Vector Vector allows to pass several numbers (integer, double or String) as parameter. Vector is enclosed between curly brackets '{ }', and values are separated by comma ','.

```
nb_pop:5
init_size:{100,200,300,400,500}
```

Usually vector are defined in whole. For example, the size of vector for *init_size* is 5 as the number of population (*nb_pop*). If the vector is a repetition of a pattern, it is also possible to define only the pattern. In this case, the pattern is automatically repeated as needed.

```
nb_pop:5
init_size:{200,100}
```

In this example, there is 5 populations, however the initial size is only specified for 2 populations. As 2 is lower than 5 a warning message will be returned and the populations will automatically have the following initial size: `init_size:{200, 100, 200, 100, 200}`. If all the values of a vector are identical, only one number is needed. In the following example, the two specifications of the initial size are equivalent:

```
nb_pop:5
init_size:{100,100,100,100,100}
init_size:100
```

2.3.2.5 External file In general, arguments are written directly after the parameter name on a single line. However, some parameters have to be given in external text file. The format file is csv (comma-separated values). The parameter value for external file is the name of the file. If the file is in the current folder, the name is sufficient but otherwise the path to the file is needed. To simplify, it's possible to use the character '*' before the name of file if it is located in the same folder as the settings file.

```
col_network:settings/matrixCol.csv
col_network:*matrixCol.csv
```

2.3.2.6 Comments It is possible to write comments in the settings file. Comments have to start with the '#' character. For multi-line comments, this character has to be used at the beginning of each line.

2.4 Output Files

CropMetaPop can produce up to 6 output files:

- 2 permanent files which are created at each simulation:
  - a file giving, for each replicate and each population, the number of each mono-locus genotype, at each marker per generation

  - a file giving the time of simulation, the warning messages and a summary of settings.

- 4 optional files:
  - a file giving, for each replicate and each population, the number of each multi-locus genotype, per generation
  - a file giving, for each replicate and each population, the number of each multi-locus haplotype, per generation
  - a file giving the list of the number of individuals who moved from one population to another through migration per generation
  - a file giving the list of the number of individuals who moved from one population to another through colonization per generation

A copy of the settings file and other external files will be created with outputs.

2.5 Minimal settings file

This section describes the parameters needed for a minimal settings file and describes the simulated model.
In CropMetaPop, one parameter is needed in every settings file:

`carr_capacity:1000`

This minimal settings file allows to perform a simulation with:

- only one population consisting of 1000 hermaphrodites individuals.
- the population evolves under random mating
- during only one generation while keeping the population size constant and equal to the carrying capacity.
- the default population has one marker with 2 alleles, initialized at 0.5.

3 Simulation

This section describes general parameters of a simulation:

3.0.0.1 generations [Integer] (Default: 1) Number of generations to perform during the simulation.

3.0.0.2 replicates [Integer] (Default:1) Number of replicates to create when running CropMetaPop. Replicates uses the same set of parameter values, but outputs may differ due to stochastic processes.

3.0.0.3 folder [String] (Default:simulation) This parameter specifies the name of the folder where all the output files are stored. This folder is created by default in the working directory but it is possible to give another location by writting this location before the name of folder. It is possible to use the '*' before the name of folder to locate this folder in the same directory as the settings file.
If the user wants to store the outputs directly in the working directory, the folder parameter has to be listed in the settings file followed by an empty argument and the argument of parameter `folder_time` will be 0. If the output files produced by CropMetaPop already exist in the given directory, they overwrite the previous ones.

3.0.0.4 folder_time [0 or 1] (Default:1) This parameter allows to specify if the name of folder contains or not the date of the simulation.

0 :the folder name does not contain the date of the simulation

1 :the folder name is dynamic, i.e. comprises the date of the simulation and the starting time with the following format: "`_yyyy_mm_dd_hh:mm:ss`" (Year_Month_Day_Hour:Minute:Second). This dynamic folder name allows to store each simulation in separate and unique folder.

3.0.0.5 seed [Integer] This parameter specifies the seed which is used to initialize the random number generator. The seed is a whole number. The seed used in one simulation is available in output file of CropMetaPop and can be used again to reproduce the same results.

3.0.0.6 Warning for Windows users! Be aware that the maximal value you can set the seed is 4294967295. Moreover, the seed does not allow to reproduce the same results between Windows and the other Operating Systems.

4 Meta-Population

This section describes general parameters concerning the metapopulation.

4.0.0.1 nb_pop [Integer] (Default:1) This parameter allows to set the number of populations. The numbering of populations begins to 0.

4.0.0.2 init_size [Integer/vector] : This parameter allows to set different population sizes for the initialization generation. If the argument is a single value, the initial population size is the same for all populations. It is possible to set an initial size for each population by providing a vector of length equal to the number of populations.
If the initial population size (parameter `init_size`) is not specified, populations are initialized with carrying capacity values.

4.0.0.3 carr_capacity [Integer/vector] : This parameter is the only parameter required in the settings file. This parameter allows to set the carrying capacities of the populations. The carrying capacity of a population is the maximal number of individuals supported by the patch (field) where the population grows. If the argument is a single value, the carrying capacity is the same for all the populations. It is possible to set the carrying capacity for each population by providing a vector of length equal to the number of populations.

```
nb_pop:2
carr_capacity:100
```

```
nb_pop:2
carr_capacity:{100,200}
```

In the first example, all populations have a carrying capacity equal to 100 individuals. In the second example, the population 0 has a carrying capacity equal to 100 individuals while the population 1 has a carrying capacity equal to 200 individuals.

4.0.0.4 nb_marker [Integer] (Default:1) : This parameter allows to set the number of markers per individual. The numbering of markers begins to 0.

4.0.0.5 nb_allele [Integer/vector] (Default:2) : This parameter allows to set the number of alleles per marker. The numbering of alleles begins to 0.

```
nb_marker:3
nb_allele:2
```

In the above example, there is 3 markers with 2 alleles for each of them.

```
nb_marker:3
nb_allele:{2,3,4}
```

In the above example, there is 3 markers. The first marker has 2 alleles, the second has 3 alleles and the third has 4 alleles.
CropMetaPop recognizes 4 forms of input format for the initialization of individual genotypes, corresponding to 4 parameters: `init_AlleleFrequency`, `init_AlleleFrequency_equal`, `init_GenotypeFrequency` and `init_GenotypeFrequency_equal`. Only one parameter has to be used in the settings file. If none of them is used, the populations are initialized with the same frequency for every allele of the same marker. This frequency is equal to one over the number of alleles per marker.

4.0.0.6 init_AlleleFrequency [External file] This parameter allows to set the initial allele frequencies of each population. This frequency may be different depending to the population. This parameter requires a text file using comma as separator (type csv). Each line of this file is organized as follows: *population, marker, frequencyAllele_0, frequencyAllele_1, ...*, with:

- *population [Integer]*: number of the population
- *marker [Integer]*: number of the marker
- *frequencyAllele_i [Double]*: frequency ranging from 0 to 1 for the allele i of the marker.

Allele frequencies have to sum to 1 for every line. The initial frequencies for all markers for all populations must be given. If the frequency of one allele is not given, it is considered equal to 0.

```
nb_pop:2
nb_marker:3
nb_allele:{2,3}
init_AlleleFrequency:*freq.csv
```

The file `freq.csv` is the following:

```
0, 0, 0.5, 0.5
0, 1, 0.2, 0.3, 0.5
0, 2, 0.2, 0.8
1, 0, 0.5, 0.5
1, 1, 0.5, 0, 0.5
1, 2, 1
```

It is possible to use the parameter *init_AlleleFrequency_equal* to set the same initial allele frequency for all populations, as described next.

4.0.0.7 init_AlleleFrequency_equal [External file] This parameter allows to set the same initial allele frequencies for each population. This parameter requires a text file using comma as separator (type csv). Each line of this file is organized as follows: *marker, frequencyAllele_0, frequencyAllele_1, ...*, with:

- *marker [Integer]*: number of the marker
- *frequencyAllele_i [Double]*: frequency range from 0 to 1 for the allele i of the marker.

Allele frequencies have to sum to 1 for every line. The initial allele frequencies for all markers must be given. If the frequency of one allele is not given, it is considered equal to 0.

```
nb_pop:2
nb_marker:3
nb_allele:{2,3}
init_AlleleFrequency_equal:*freq.csv
```

The file `freq.csv` is the following:

```
0, 0.5, 0.5
1, 0.2, 0.3, 0.5
2, 1
```

4.0.0.8 init_GenotypeFrequency [External file] This parameter allows to set the initial multi-locus genotype frequencies for each population. This frequency may be different depending on the population. The composition in terms of multi-locus genotypes can also be different depending on the population. This parameter requires a text file using comma as separator (type csv). Each line of this file is built as follows: *population, genotypeMarker_1, genotypeMarker_2, ..., frequency*, with:

- *population [Integer]*: number of the population
- *genotypeMarker_i [String]*: The genotype of the marker i must be writen as *allele/allele*.
- *frequency [Double]*: frequency between 0 and 1 of the multi-locus genotype in the population.

In this file, all the markers have to be filled. All the multi-locus genotypes per population have to be provided. Multi-locus frequencies have to sum to 1 per population.

```
nb_pop:3
nb_marker:2
nb_allele:{2,3}
init_GenotypeFrequency:*freq.csv
```

The file `freq.csv` is the following:

```
0, 1/1, 1/1, 0.5
0, 1/0, 0/0, 0.5
1, 0/0, 0/0, 0.3
1, 1/0, 1/2, 0.5
1, 0/0, 1/1, 0.2
2, 1/1, 0/0, 1
```

It is possible to use the parameter *init_GenotypeFrequency_equal* if all the populations have the same initial genotypes, as described next.

4.0.0.9 init_GenotypeFrequency_equal [External file] This parameter allows to set the multi-locus genotype frequencies per population during initialization. This parameter requires a text file using comma as separator (type csv). Each line of this file is built as follows: *genotypeMarker_0, genotypeMarker_1, . . . , frequency*, with:

- *genotypeMarker_i [String]*: The genotype of the marker i must be write as *allele/allele*.
- *frequency [Double]*: frequency between 0 and 1 of the multi-locus genotype.

In this file, all the markers have to be filled. The multi-locus genotypes frequencies has to sum to 1.

```
nb_pop:2
nb_marker:3
nb_allele:{2,3}
init_GenotypeFrequency_equal:*freq.csv
```

The file `freq.csv` is of the following shape:

```
1/1, 1/1, 0.25
0/0, 0/2, 0.25
0/0, 1/1, 0.5
```

4.0.0.10 geneticMap [External file] This file allows to give the position of each marker on chromosome. This parameter requires a text file using comma as separator (type csv). Each line of this file is built as follows: *marker, chromosome, distance*, with:

- *marker [Integer]*: number of marker
- *chromosome [Integer]*: number of chromosome
- *distance [Double]*: This distance between the extremity of chromosome and the marker, in morgan.

All the markers have to be filled. Moreover, markers have to be ordered by distance.

```
geneticMap:*genetic.csv
```

The file `genetic.csv` with the configuration file of the previous example is of the following shape:

```
0, 1, 0.5
1, 2, 1
3, 2, 2.5
```

The following formula is used to calculate the recombination rate between pairs of markers (Haldane) thanks to the disctance between two adjacent markers (D):

$$recombination = \frac{1}{2}(1 - e^{-2D})$$

If the parameter *geneticMap* is not given, each marker is considered as independent and the recombination rate is fixed to 0.5.

5 Life Cycle

CropMetaPop relies on SimuPop, a discrete generation-based simulator. This means that a single individual undergoes only once during the life cycle without overlapping generations. Depending on the parameterization of the settings file some processes can be skipped. The order of the different processes of the life cycle is fixed. A simulation starts with 'Breeding', i.e. only adults are present at the initialization of the simulation. A new life cycle starts again at the end of each generation up to the last generation, i.e. before the last generation a new life cycle starts with "Breeding" and comes after 'Output':

- **Breeding**: adult plants produce new seeds for the starting generation. Selection and mutation are taken into account at this stage. After reproduction, the adult plants are removed and the new seeds constitute the new generation.
- **Extinction**: Due to stochastic events populations may extinct.
- **Seed circulation**: A seed lot sample can be tranferred from one or several populations to another one.
- **Regulation**: After these three previous steps, the number of seeds may be too large compared to the carrying capacity. This regulation stage allows to control the population sizes. The remaining seeds become adult plants that will be used to create the next generation.
- **Output**: Different options are available to customize the output files after the simulations.

5.1 Breeding

This stage performs mating and breeding to produce the new offspring generation. After that step, the adult plants are removed and the population is composed of the offsprings. CropMetapop allows the evolution of population size as the number of offsprings is the produce between the parameter *fecundity* and the number of adults ($number\_adult$) ($number\_offspring = number\_adult \times fecundity$).

5.1.0.1 fecundity [Double] (Default:1) This parameter sets the average number of offspring produced per individual. Default value for *fecundity* is 1 corresponding to a constant population size.

```
fecundity:1
```

5.1.1 Selection

In this version of CropMetaPop, local selection is modeled as a combination of soft selection and selection at the population level. Therefore, selection influences the breeding at two levels. At the population level, the effective number seeds produced in a population is adjusted with the mean fitness of the adults of this population ($number\_seed = number\_offspring \times mean(fitness)$). At the individual level, the parents of each offspring are drawn among all available parents within the population depending on their individual fitness, i.e. the higher the individual fitness, the higher the probability to be chosen.

CropMetaPop simulates local selection in different populations using two possibilities: either the same genotype may have different fitness values depending on the population (in this

case, use the parameter *fitness*) or each population has its own optimum fitness (in this case, use the parameter *fitness_equal*).

There are two types of markers:

- neutral markers do not influence the individual fitness
- gene markers influence the individual fitness. Defining several gene markers allows to model a quantitative trait. The allelic effects at each locus must be set explicitly in the parameters *fitness* or *fitness_equal*

5.1.1.1 fitness_equal [External file] This parameter allows to set a fitness value for each genotype at each selective marker. Each line of this file is built as *marker, genotype, value*. Marker in this file is automatically considered in gene region. Default individual fitness value is equal to the optimum of the population.

- *marker [Integer]*: marker's identifier
- *genotype [String]*: The genotype must be written as *Allele/Allele*.
- *value [Double]*: individual fitness value for the considered genotype at the considered marker. Individual fitness ranges from 0 to 1, by representing the relative reproductive success of an individual carrying this genotype for the considered marker.

```
nb_pop:3
fitness_equal:*fit.csv
```

With the `fit.csv` file being of the following shape:

```
0, 1/1, 0.5
0, 1/0, 0.2
0, 0/0, 0.2
1, 1/1, 0.1
1, 1/0, 0.8
1, 0/0, 0.3
```

From this file, the absolute fitness (fit_abs) of one individual is calculated as the mean of fitness values of its genotype at each marker in gene region. In the above example, the markers 0 and 1 are gene markers. An individual with a genotype 1/1 for the marker 0 and the genotype 1/0 for the marker 1 has $fit\_abs = (0.5 + 0.8)/2 = 0.65$

5.1.1.2 fitness [External file] This parameter allows to set the fitness values for each genotype at each gene marker in each population. Each line of this file is built as *population, marker, genotype, value*. Marker in this file is automatically considered as gene marker. Default fitness value is fixed to the optimum of the population.

- *population [Integer]*: number of number
- *marker [Integer]*: marker's ID
- *genotype [String]*: genotype must be written as *numberOfAllele1/numberOfAllele2*.
- *value [Double]*: selective value between 0 and 1 for the genotype of the marker in this population. It describes the reproductive success of an individual with this genotype at this marker.

```
fitness:*fit.csv

0, 0, 1/1, 0.5
1, 1, 1/0, 0.8
2, 0, 0/0, 0.9
2, 1, 0/1, 0.85
```

5.1.1.3 optimum [Double/vector] (Default:1) This parameter specifies the optimum of fitness for each population. If the argument is a single value, the optimum is the same for all the populations. By passing a vector of optima, it is possible to assign an optimum value of fitness for each population.

```
nb_pop:2
optimum:0.8

nb_pop:2
optimum:{0.2,0.8}
```

In the first example, all the populations have an optimum of 0.8. In the second example, population 0 has an optimum of 0.2 whereas population 1 has an optimum of 0.8.

Finally, the fitness of one individual is given by:

$$fitness = e^{-\frac{1}{2} \times (fit\_abs - optimumPopulation)^2}$$

The closest to the optimum of the population is the absolute fitness of an individual, the highest is the final fitness.

5.1.2 Mating System

In our model, individuals are hermaphrodite and individuals have two possibilities for mating:

- *Selfing*: one parent from a given population is randomly assigned to self fertilize and create a new individual
- *Random mating*: two parents from the same population are randomly assigned to create a new individual.

The proportion of selfing in the simulation is determined by the parameter *percentSelf*

5.1.2.1 percentSelf [Double] (Default:0) This parameter allows to determine the proportion of offspring which will be created by selfing. It must be between 0 and 1. When this parameter is equal to 1, all the offsprings are created by selfing whereas when it is equal to 0, all the offsprings are created by random mating.

```
percentSelf:0.8
```

In this case, 80% of individuals are created by selfing and 20% by random mating.

5.1.3 Mutation

The mutation model used in CropMetaPop is a K-allele model. In this model, each allele has the same probability to mutate into another allele. The rate of mutation is defined by the parameter *mut_rate*.

5.1.3.1 mut_rate [Double/vector] (Default:0) This parameter specifies the mutation rate by locus and generation. If the argument is a single value the mutation rate for all loci is the same. By passing a matrix of mutation rates it is possible to set the mutation rate for each single locus individually. By default no mutations occur. The mutation rate ranges from 0 to 1.

```
nb_marker:2
mut_rate:0.01

nb_marker:2
mut_rate:{0.01,0.05}
```

In the first example, the mutation rate for both markers is 0.01. However, in the second example, the marker 0 has a mutation rate equal to 0.01 and equal to 0.05 for the marker 1.

5.2 Extinction

This event allows to randomly make population extinct. When a population goes extinct, all individuals of the population are removed and the patch becomes empty.

5.2.0.1 ext_rate [Double/vector] (default: 0) The extinction rate corresponds to the probability of a population to be extinct at each generation. It is specified by the parameter *ext_rate*. This event is skipped when the extinction rate is equal to zero. When this argument is a single value, all the populations have the same extinction rate. It is possible to set the extinction probability for each single population by passing a vector. The extinction rate ranges from 0 to 1.

```
nb_pop:2
ext_rate:0.2

nb_pop:2
ext_rate:{0.2,0.3}
```

In the first example, the extinction rate for both populations is 0.2. Whereas in the second example, the extinction rate is 0.2 for population 0 and 0.3 for population 1.

5.3 Seed Circulation

Seed circulation is a two step process: the first one is composed of colonization and/or migration models, and the second one by a seed transfer model.

The colonization and migration models define the rules of seed circulation and the seed transfer model defines the quantity of seed in circulation.

Colonization occurs only to fill an empty patch while migration can occur to fill an empty or not empty patch. Networks of seed circulation are similarly defined for colonization and migration. Only the name of the parameters are different (*col_param* and *migr_param* respectively).

5.3.1 Colonization model

Colonization accounts for cases when a farmer has lost his population due to an extinction event for instance and wants to obtain a new seed lot to replace it. It is then possible for him to obtain seeds from his social network through a direct neighbor who still grows the population.

5.3.1.1 Required parameters Along with the colonization network that needs to be defined (*cf* SECTION 5.3.1.2), you might want to make sure the `fecundity` parameter (*cf* SECTION 5.1.0.1) default value fits your need. If you are using colonization_with_excess, `fecundity` should be set above 1 for colonisation events to happen.

5.3.1.2 col_network [0,1,2,3,4,5,6 or external file] (Default 0) This parameter allows to choose the network topology of seed circulation for colonization, i.e the social network linking farmers who possibly can provide seed lots. A network topology is defined by a set of nodes (populations) and a set of edges (social links allowing seed circulation between two populations). Two ways are available to define the network topology: i. when the parameter ranges from 1 to 6, different network models can be selected; ii. when it is an external file, it is possible to directly provide a matrix of connection including colonization rates or an adjacency matrix.

0. **No colonization**: populations are not connected with each other (empty network).
1. **Stepping stone 1D model**: each population is connected to 2 neighbors (chain).
2. **Stepping stone 2D model**: each population is connected to 4 neighbors (grid). With this colonization network, **the number of populations needs to have an integer square root**.
3. **Island model**: all the populations are connected with each other (complete network).
4. **Erdős-Rényi model**: a random graph model where each pair of populations has the same probability to be connected (Erdős & Rényi, 1959).
5. **Community model**:a random graph model based on the stochastic block model defined by at least two clusters and two probability of connection (within and between cluster). The within-cluster probability needs to be higher than the between-cluster probability to obtain a community network topopology (Snijders & Nowicki, 1997).
6. **Barabási-Albert model**:it is a random graph model based on preferential attachment algorithm (Albert & Barabási, 2002). It is a discrete time step model and in each time step a single vertex is added. We start with a single vertex and no edges in the first time step. Then we add one vertex in each time step and the new vertex initiates some edges to old vertices. The probability that an old vertex is chosen is given by:$P[i]\ k[i]^{\alpha}$ where k[i] is the in-degree of vertex i in the current time step (more precisely the number of adjacent edges of i which were not initiated by i itself) and α is a parameter given by the power arguments.

Model	Additional Parameters
0: No colonization	
1: Stepping stone 1D model	*col_rate*
2: Stepping stone 2D model	*col_rate*
3: Island model	*col_rate*, *col_directed*
4: Erdős-Rényi model	*col_rate*, *col_directed*, *col_nb_edge*
5: Community model	*col_rate*, *col_directed*, *col_nb_cluster*, *col_prob_intra*, *col_prob_inter*
6: Barabási-Albert model	*col_rate*, *col_directed*, *col_power*

5.3.1.3 col_rate [Double] (Default:0) This parameter defines the colonization rate between all pairs of populations of the metapopulation. The value of this parameter ranges from 0 to 1. The same colonization rate is applied to the network by using this parameter. It is also possible to choose different colonization rates by directly loading a matrix of connection specifying each colonization rate using an external file (see below). If this parameter is not filled, then populations are considered as independent, corresponding to a situation without colonization (equivalent to the option `col_network=0`).

5.3.1.4 col_directed [0,1] (Default:0) This parameter defines if the created network will directed.

0: The network won't be directed

1: The network will be directed

5.3.1.5 col_nb_edge [Integer] (Default:0) This parameter sets the number of edges present in a network defined through a Erdős-Rényi model. If this parameter is not filled for an Erdős-Rényi model, then the number of edge is set to 0 and populations are considered as independent, corresponding to a situation without colonization (equivalent to the option `col_network=0`).

5.3.1.6 col_nb_cluster [Integer] (default:1) This parameter allows to set the number of clusters present a network topology obtained through the Community model.

5.3.1.7 col_prob_intra [Double] (Default:1) This parameter allows to set the within-cluster probability of connection in the Community model, i.e. the probability for two populations from the same cluster to be connected together. The value of this parameter ranges from 0 to 1.

5.3.1.8 col_prob_inter [Double] (Defaut:1) This parameter allows to set the between-cluster probability of connection in the Community model, i.e. the probability for two populations from two different clusters to be connected together. The value of this parameter ranges from 0 to 1.

5.3.1.9 col_power [Double] (Default:1) This parameter allows to set the power for the Barabási-Albert.

It is also possible to give a self-defined connection matrix of desired network with an external file. Two ways are possible to define such colonization model: i. the external file is a matrix filled with 0 and 1, corresponding to an adjacency matrix and the *col_rate* option is used to defined the general colonization rate of the network; ii. the external file is filled with values ranging from 0 to 1, corresponding to local colonization rate.

```
col_network:*matrix.csv
col_rate:0.2

0, 0, 1
1, 0, 1
0, 1, 0
```

or

```
col_network:*matrix.csv

0, 0, 0.2
0.2, 0, 0.2
0, 0.2, 0
```

These 2 examples give the same results corresponding to the two above mentioned possibilities respectively.

5.3.2 Migration model

Migration accounts for cases when a farmer introduces a new source of seed into his population. It is then possible for him to obtain additional seeds from his social network through a direct neighbor who grows the population.

5.3.2.1 Required parameters: Along with the migration network that needs to be defined (*cf* SECTION 5.3.2.2), you might want to make sure the `fecundity` parameter (*cf* SECTION 5.1.0.1) default value fits your need. If you are using migration_with_excess, `fecundity` should be set above 1 for colonisation events to happen.
The `migr_replace` parameter (*cf* SECTION 5.3.3.4) default value must also be replaced in your configuration file in order for migration events to happen.

5.3.2.2 migr_network [0,1,2,3,4,5,6 or external file] (Default 0) This parameter allows to choose the network topology of seed circulation for migration, i.e the social network linking farmers who possibly can provide seed lots. A network topology is defined by a set of nodes (populations) and a set of edges (social links allowing seed circulation between two populations). Two ways are available to define the network topology: i. when the parameter ranges from 1 to 6, different network models can be selected; ii. when it is an external file, it is possible to directly provide a matrix of connection including migration rates or an adjacency matrix.

0. **No migration**: populations are not connected with each other (empty network).
1. **Stepping stone 1D model**: each population is connected to 2 neighbors (chain).
2. **Stepping stone 2D model**: each population is connected to 4 neighbors (grid). With this migration network, **the number of populations needs to have an integer square root**.
3. **Island model**: all the populations are connected with each other (complete network).
4. **Erdős-Rényi model**: a random graph model where each pair of populations has the same probability to be connected[3].
5. **Community model**:a random graph model based on the stochastic block model defined by at least two clusters and two probability of connection (within and between cluster). The within-cluster probability needs to be higher than the between-cluster probability to obtain a community network topopology[5].
6. **Barabási-Albert model**:it is a random graph model based on preferential attachment algorithm[1, 2]. It is a discrete time step model and in each time step a single vertex is added. We start with a single vertex and no edges in the first time step. Then we add one vertex in each time step and the new vertex initiates some edges to old vertices. The

probability that an old vertex is chosen is given by:$P[i]$ $k[i]^{\alpha}$ where k[i] is the in-degree of vertex i in the current time step (more precisely the number of adjacent edges of i which were not initiated by i itself) and α is a parameter given by the power arguments.

Model	Additional Parameters
0: No migration	
1: Stepping stone 1D model	*migr_rate*
2: Stepping stone 2D model	*migr_rate*
3: Island model	*migr_rate*, *migr_directed*
4: Erdős-Rényi model	*migr_rate*, *migr_directed*, *migr_nb_edge*
5: Community model	*migr_rate*, *migr_directed*, *migr_nb_cluster*, *migr_prob_intra*, *migr_prob_inter*
6: Barabási-Albert model	*migr_rate*, *migr_directed*, *migr_power*

5.3.2.3 migr_rate [Double] (Default:0) This parameter defines the migration rate between all pairs of populations of the metapopulation. The value of this parameter ranges from 0 to 1. The same migration rate is applied to the network by using this parameter. It is also possible to choose different migration rates by directly loading a matrix of connection specifying each migration rate using an external file (see below). If this parameter is not filled, then populations are considered as independent, corresponding to a situation without migration (equivalent to the option `migr_network=0`).

5.3.2.4 migr_directed [0,1] (Default:0) This parameter defines if the created network will directed.

0: The network won't be directed

1: The network will be directed

5.3.2.5 migr_nb_edge [Integer] (Default:0) This parameter sets the number of edges present in a network defined through a Erdős-Rényi model. If this parameter is not filled for an Erdős-Rényi model, then the number of edge is set to 0 and populations are considered as independent, corresponding to a situation without migration (equivalent to the option `migr_network=0`).

5.3.2.6 migr_nb_cluster [Integer] (default:1) This parameter allows to set the number of clusters present a network topology obtained through the Community model.

5.3.2.7 migr_prob_intra [Double] (Default:1) This parameter allows to set the within-cluster probability of connection in the Community model, i.e. the probability for two populations from the same cluster to be connected together. The value of this parameter ranges from 0 to 1.

5.3.2.8 migr_prob_inter [Double] (Defaut:1) This parameter allows to set the between-cluster probability of connection in the Community model, i.e. the probability for two populations from two different clusters to be connected together. The value of this parameter ranges from 0 to 1.

5.3.2.9 migr_power [Double] (Default:1) This parameter allows to set the power for the Barabási-Albert.

It is also possible to give a self-defined connection matrix of desired network with an external file. Two ways are possible to define such migration model: i. the external file is a matrix filled with 0 and 1, corresponding to an adjacency matrix and the *migr_rate* option is used to defined the general migration rate of the network; ii. the external file is filled with values ranging from 0 to 1, corresponding to local migration rate.

```
migr_network:*matrix.csv
migr_rate:0.2

0, 0, 1
1, 0, 1
0, 1, 0
```

or

```
migr_network:*matrix.csv

0, 0, 0.2
0.2, 0, 0.2
0, 0.2, 0
```

These 2 examples give the same results corresponding to the two above mentioned possibilities respectively.

5.3.3 Seed transfert model

The seed transfer model allows to set the parameters necessary to define the rules to quantify the seed in circulation during seed circulation events.

5.3.3.1 col_transfer_model & migr_transfer_model [String] (Default: 'excess') These parameters determine the seed transfer model used to calculate how many seeds will be transferred among populations during the colonization event and the migration event respectively.

- **excess**: this parameter value corresponds to the situation when a farmer provides seed to another farmer only if his population produces more seeds than its carrying capacity. Furthermore, the number of outgoing seeds is adjusted with the carrying capacity of the recipient population.
  The number of seed transferred is thus calculated as the minimum between the donor capacity to provide seed and the recipient capacity to receive seeds. If a farmer provides seeds to several farmers, the global capacity to provide seeds of the donor is equally distributed among the recipients. Likewise, if a population receives seeds from several populations, then the total amount of seeds received is equally distributed among the seed sources.

- **friendly**: this parameter value corresponds to the situation when a farmer provides seed to another farmer even if his population don't produces enough seed for sowing all the field.

5.3.3.2 col_from_one & migr_from_one [0,1] (Default:0) These parameters allow to reduce the send of seed in colonisation or migration from only one population.

0 : A population can received seeds from several populations in one colonisation or migration event.

1 : A population can received seed from only one population in one colonisation or migration event.

5.3.3.3 migr_carrying [0,1] (Default:0) This parameter allows to control if a population can receive migrant seeds when it isn't at its carrying capacity.

0 : The recipient population has to be at its carrying capacity to receive seeds by migration.

1 : The recipient population does not need to be at its carrying capacity to receive seeds by migration. In this case, the migrant seeds complete the local seed without exceeding the carrying capacity.

5.3.3.4 migr_replace [Double/vector] (Default:0) This parameter allows to set the replacement rate for each population. This corresponds to the percentage of local seeds that will be replaced by the migrant seeds. If *migr_replace* is a single value, then it is the same replacement rate for all the populations. If it is a vector, then a replacement rate is defined for each population. The value must be between 0 and 1.

5.3.3.5 col_keepRate [Double/vector] (Default:0.5) & migr_keepRate [Double/vector] (Default:0.5) These parameters allows to set the minimum proportion that the population of origin of the seeds keeps instead of sharing with requesting populations. This parameter can be used both for colonisation and for migration.

They are only used when the transfer model (see SECTION 5.3.3.1) is set to friendly. When it is, make sure to redefine it as you need.

5.4 Regulation

This event allows to limit the temporary population size obtained after the previous steps to its *carrying capacity*.

After this step, the seeds become adult individuals (plants), starting a new life cycle by reproducing during the breeding process.

5.5 Outputs

At the end of each life cycle, i.e. one generation, information corresponding to the mono-locus genotypes of individuals are stored to be written in the result file after the simulation. This information is stored at every step determined by the parameter *step*.

5.5.0.1 step [Integer] (Default:1) This parameter corresponds to the frequency to store individual information into output files.

```
step:5
```

In this example, data is stored every 5 generations.

5.5.0.2 outputs [String/vector] This parameter defines optional results files which will be created after the simulation.

- **Genotype**: creates a file with the number of each unique multi-locus genotype per population, for all the requested generations and all the replicates. The time to write this file is proportional to the number of markers used in the simulation.
- **Haplotype**: creates a file with the number of each haplotype per population for all the requested generations and all the replicates.
- **Seed_transfert**: creates two files summarizing the different seed transfer events and seed quantity at each generation for all the populations and replicate. One file corresponds to the colonization events and the second to the migration events.

The following exammples are presented for:

```
nb_pop:2
nb_allele:2
nb_marker:1
```

Mono-locus genotype result example:

Replicate	Population	Marker	Genotype	Gen 0	Gen 1	Gen 2	...
0	0	0	0/0	25	28	31	...
0	0	0	0/1	49	20	19	...
0	1	1	1/1	26	28	31	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱

Multi-locus genotype result example:

Replicate	Population	Marker 0	Marker 1	Gen 0	Gen 1	Gen 2	...
0	0	0/0	0/0	25	28	31	...
0	0	0/0	1/0	22	20	19	...
0	1	0/0	0/0	26	28	31	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱

Haplotype result example:

Replicate	Population	Marker 0	Marker 1	Gen 0	Gen 1	Gen 2	...
0	0	0	0	25	28	31	...
0	0	0	1	22	20	19	...
0	1	0	0	26	28	31	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱

Colonization result example:

Replicate	Source	Target	Gen 0	Gen 1	Gen 2	...
0	0	1	25	0	31	...
0	0	2	22	0	0	...
0	1	0	0	28	31	...
⋮	⋮	⋮	⋮	⋮	⋮	⋱

```
outputs:Haplotype,Seed_transfert
```

5.5.0.3 separate_replicate [0,1] (Default:0) This parameter allows to separate the results by replicates.

0 : All results are in a same file

1 : A different file is created for each replicate.

References

[1] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1):47–97, January 2002. Publisher: American Physical Society.

[2] Albert-László Barabási and Réka Albert. Emergence of Scaling in Random Networks. *Science*, 286(5439):509–512, October 1999.

[3] Paul Erdős and Alfred Rényi. On random graphs. *Publicationes mathematicae*, 6(26):290–297, 1959.

[4] Bo Peng and Marek Kimmel. simuPOP: a forward-time population genetics simulation environment. *Bioinformatics*, 21(18):3686–3687, September 2005.

[5] Tom A.B. Snijders and Krzysztof Nowicki. Estimation and Prediction for Stochastic Blockmodels for Graphs with Latent Block Structure. *Journal of Classification*, 14(1):75–100, January 1997.