Chapter 3
Monte Carlo Simulations to Model the Behaviour of Agricultural Pests and Their Natural Enemies

Eric Wajnberg

Abstract Over the last decades, diverse modelling approaches have been used to understand insect behaviour and population dynamics in agricultural landscapes and to improve our ability to manage crop pests. When there are too many parameters used to define insect behavioural reproductive strategies and environmental characteristics, standard modelling tools become mathematically intractable, and so-called Monte Carlo computer-assisted simulation methods can be developed instead. Most of the time, simulations are done in spatially defined environments; hence, these simulations are usually said to be 'spatially explicit'. Such approaches can be coupled with numerical tools to find the parameters that optimise some pre-defined objective criteria, such as fitness output or pest control efficacy. Through examples, this chapter will present these methods and how they can help us to understand insect behaviour and their populations, and thus to potentially optimise pest control strategies.

Keywords Monte Carlo · Simulation · Pest control · Natural enemies · Stochasticity · Insect behaviour · Biological control

3.1 Introduction

Theoretical models developed to understand animal, and especially insect behaviour over the last decades, have followed a variety of approaches. Mostly these models are based either on the aim to understand the demographic trajectories – in both time and space – of the species studied or on trying to find optimal reproductive strategies adopted by individuals in different environmental situations (Godfray,

E. Wajnberg (✉)
INRAE, Sophia Antipolis Cedex, France

INRIA – Projet Hephaistos, Sophia Antipolis Cedex, France
e-mail: eric.wajnberg@inrae.fr

R. A. Moral, W. A. C. Godoy (eds.), *Modelling Insect Populations in Agricultural Landscapes*, Entomology in Focus 8, https://doi.org/10.1007/978-3-031-43098-5_3

1994; Wajnberg et al., 2008). In this last case, the modelling techniques used were borrowed from optimisation theory (Houston & McNamara, 1999).

Most models are sufficiently simple and tractable to be solved using standard optimisation tools. However, in an increasing number of situations, the problems that need to be addressed are of an increasing complexity and are difficult to be solved analytically (Hoffmeister & Wajnberg, 2008). For example, these models can now take into account changes in the state of the animal during its resource foraging time using so-called stochastic dynamic programming models (SDPs; Clark & Mangel, 2000). In addition, interactions with competitors that are also trying to maximise their reproductive strategy can now be considered using so-called game theoretical models (Maynard-Smith, 1982).

For an increasing number of questions, problems are becoming far too complicated to be solved with these modelling techniques, and other, computer-assisted tools must be used instead. These models are based on so-called Monte Carlo simulation approaches. These are currently the only powerful tool available to tackle problems in which we need to consider situations defined by many different parameters (and/or their interactions), and especially if stochasticity (e.g. variation in the environment in which the simulated animals forage for resources) needs to be considered. The aim of this chapter is to present these simulation methods, to see how they can be implemented in computers, how their results can be analysed, and how optimised solutions can be identified.

Monte Carlo methods were initially developed in the 1940s, when the first fast computers became available. They are based on repeated random sampling to collect numerical results. Such simulation techniques are frequently used in a variety of fields, including physics (e.g. to study interacting particle systems), chemistry (e.g. to study molecule interactions), engineering (e.g. to study fluid dynamics or understanding variation in microelectronic circuits), and climate change dynamics. More recently, such methods started to be used in the field of ecology (Giró et al., 1985, 1986). The name 'Monte Carlo' was coined by the physicist Nicholas Metropolis to refer to the Monte Carlo casino in Monaco in which a significant amount of numbers is continuously drawn randomly (Metropolis, 1987).

In such models, the trajectory of each individual is followed in time and/or in space, so these models are usually called individual-based models (i.e. 'IBM') and, in the usual terminology, individuals are sometimes called 'agents', so these models are also sometimes called 'multi-agent models' or 'multi-agent-based simulation' (i.e. 'MABS'). Despite being usually conceptually and algorithmically simple, the computational cost associated with Monte Carlo simulations can be staggeringly high since getting accurate numerical outputs usually requires many replicates to be run. Hence, these methods became progressively more popular with the increasing availability of powerful computers and especially with the capability nowadays to access to local or even worldwide computer grids enabling the running of several replicates on different computers at the same time (see below).

Of course, like any other modelling tool, the process being simulated with these Monte Carlo methods is a simplification of the real situation we are endeavouring to understand. However, since many rules can be added to the simulation framework, the obtained results are generally closer to reality than those produced by other modelling approaches.

3.2 An 'Appetiser'

In this section, I present a simple didactic example to explain the general framework used to build a Monte Carlo simulation model in the field of ecology. The idea is to simulate the behaviour of a single parasitoid female foraging for hosts distributed in patches in a 2D space. The space is a 500×500 cell grid and the location of the hosts over the grid is drawn randomly using two steps. The locations of host patches are first randomly drawn all over the grid. It was arbitrarily decided that the number of patches represents 4% of the total number of hosts present in the grid. Each patch contains the same number of hosts, whose location is then drawn using a Normal distribution centred on the location on the patch each host belongs to and with a standard deviation (SD) of 30 cells. This produces reasonable levels of host aggregation (see Fig. 3.1 for an example).

A single parasitoid female is 'released' in the centre of the grid and moves following a discrete time process. At each time step, a linear speed is drawn from a Normal distribution with a mean of 5.0 and a SD of 2.0 cells, and an angular speed (direction) is also drawn from a Normal distribution with a mean equal to the angular speed used in the previous time step and a SD of 0.9424 radian (i.e. 54.0 degrees). For the first time step, the mean direction is drawn randomly between 0.0 and 2π. During its walking process, the female can perceive the nearest unattacked host from a distance of 80 cells (i.e. reactive distance; Roitberg, 1985; Bruins et al., 1994) and, if a host is perceived, the angular speed in the next time step is in the

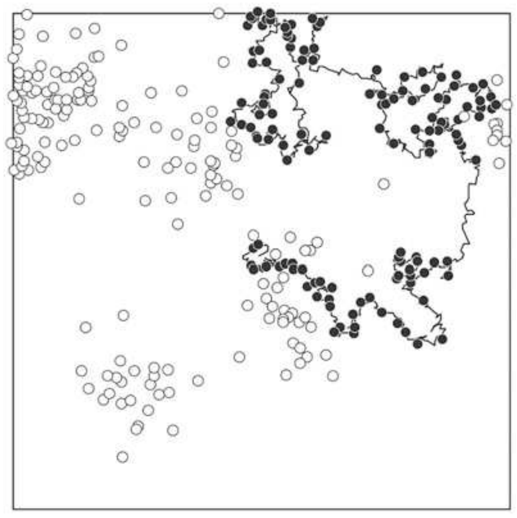
Fig. 3.1 An example of the simulated walking behaviour of a single parasitoid female foraging for hosts having an aggregated distribution in a 2D space; 300 hosts are available. The female is released in the centre of the grid and the simulation stops when the female reaches the border of the grid. White and black circles represent unattacked and attacked hosts, respectively

direction of this nearest host. Following this process, if the female gets closer than 5 cells to this targeted host, its location becomes this host location and the host is considered to be attacked. Finally, the simulation stops when the female reaches the border of the grid. Figure 3.2 gives the flowchart of the entire simulation process and Fig. 3.1 provides an example of the corresponding walking pattern obtained when 300 hosts are available.

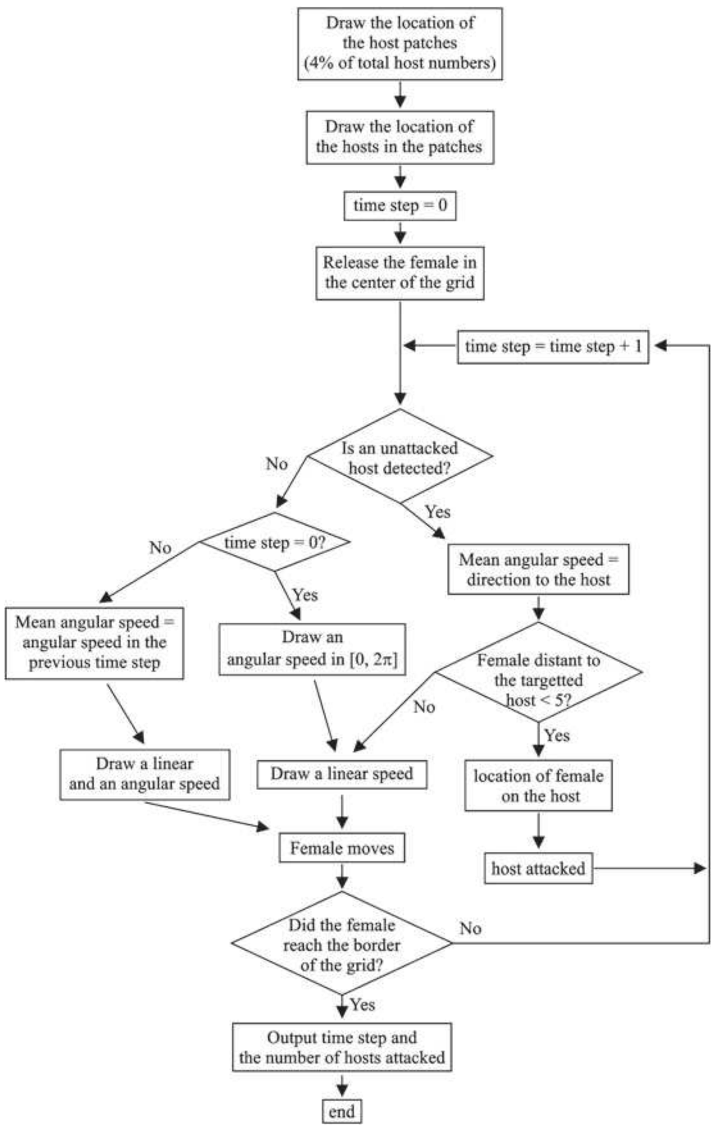
Draw the location of the host patches (4% of total host numbers)
Draw the location of the hosts in the patches
time step = 0
Release the female in the center of the grid
time step = time step + 1
Is an unattacked host detected?
No
Yes
time step = 0?
No
Yes
Mean angular speed = direction to the host
Mean angular speed = angular speed in the previous time step
Draw an angular speed in [0, 2π]
Female distant to the targetted host < 5?
No
Yes
Draw a linear and an angular speed
Draw a linear speed
location of female on the host
Female moves
host attacked
Did the female reach the border of the grid?
No
Yes
Output time step and the number of hosts attacked
end

Fig. 3.2 Flowchart of a Monte Carlo simulation model used as an example to simulate the walking behaviour of a single parasitoid female foraging for hosts exhibiting an aggregated distribution in a 2D space

After running each simulation, we can compute – as an estimation of the ability of the foraging parasitoid female to produce progeny (i.e. its fitness) – the number of hosts discovered and attacked per time unit. The model specifically depends on two important parameters defining the walking strategy to the simulated individual: (1) the mean linear speed defining the Normal distribution in which the distance travelled at each time step is randomly drawn, and (2) the SD of the angular speed defining the Normal distribution in which the direction of the animal is also drawn at each time step. The smaller this SD, the more the animal walks in a straight line.

This model was run with different mean linear speeds, ranging from 5.0 to 20.0, with a step of 1.0 cell, and different SD of the angular speed, from 0.0 to 0.6, with a step of 0.05 radian. In each case, 200 replicates were run, and the average values obtained are shown in Fig. 3.3. As we can see, there are some intermediate values of the mean walking speed that maximise the number of hosts a foraging female discovers and attacks per time unit. This makes sense since females walking slowly will take more time to find hosts to exploit, while females walking more rapidly will miss hosts, losing foraging time. Also, for the conditions used in the simulation, smaller SD for the angular speeds (i.e. walking more in a straighter line) leads the females to increase their rate of host attack efficiency. As is shown in Fig. 3.3, there

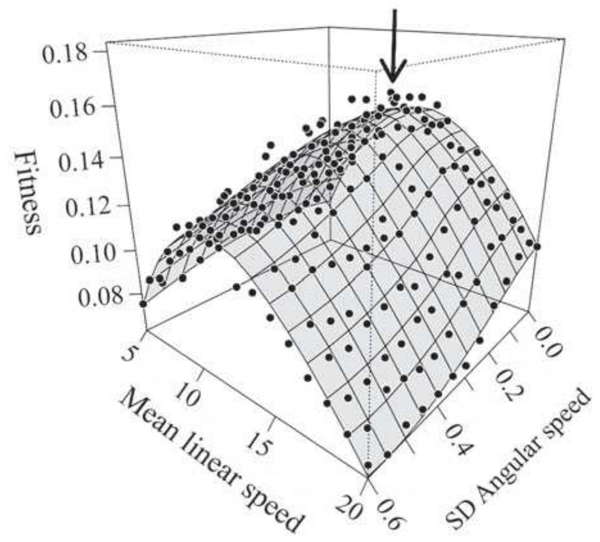

Fig. 3.3 Average fitness (i.e. number of hosts discovered and attacked per time unit), obtained by running the simulation model presented in Figs. 3.1 and 3.2, 200 times each for different values of the mean linear speed (expressed in cell unit) and the SD of the angular speed (expressed in radian). The grey surface is a fitted local regression. The arrow shows the value of the parameters (i.e. linear speed = 12.29 cells; SD of angular speed = 0.13 radian) that maximises the overall fitness output of the simulated females, as computed with a genetic algorithm (see the explanation in the text below)

is a way to find the value of these two parameters that maximises the number of attacks per time unit. This will be presented later in this chapter.

As explained above, this is a simple example to present how a Monte Carlo simulation can be conceived and how the results can be presented and discussed. More generally, the simulation framework usually depends on several (usually more than two) parameters that are called ‘state parameters’ (here the mean linear speed and the SD of the angular speed). These parameters are then combined into the simulation framework to produce a so-called objective function that produces an output criterion to be optimised (here the number of hosts discovered and attacked per time unit). Depending on the scientific problem addressed, this can be the number of progeny produced, the pest control ability of a biological control agent, the economic productivity/profitability of a crop, etc. (see, e.g. Plouvier & Wajnberg, 2018). Figure 3.4 gives a diagrammatic representation of a possible general framework of such a process.

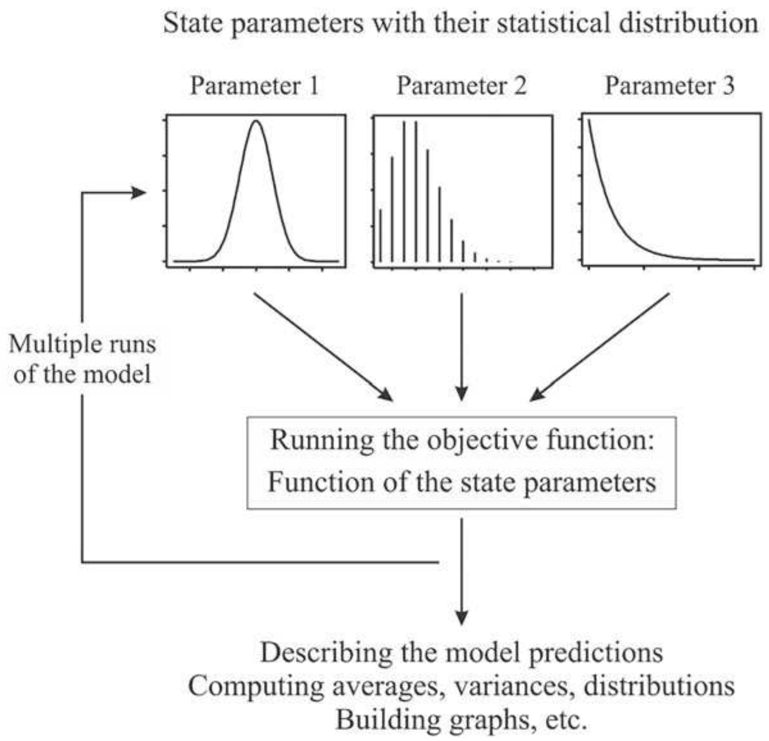
State parameters with their statistical distribution
Parameter 1
Parameter 2
Parameter 3
Multiple runs of the model
Running the objective function: Function of the state parameters
Describing the model predictions
Computing averages, variances, distributions
Building graphs, etc.

Fig. 3.4 Diagrammatic representation of the development of a Monte Carlo simulation process. In this example, the simulation model is based on three parameters that have different distributions (from left to right: Normal, Poisson, and Exponential). Randomly drawn values of these three parameters are used in the objective function simulating the situation studied, leading to produce an output criterion, and the process is replicated multiple times to collect average results, with their variance, to produce graphs, etc., and thus to understand the predictions of the model

3.3 Random Number Generators

Monte Carlo simulations are similar to ‘random experimentations’. To run each simulation, as this is the case in the example described above, we need to draw a large amount of random numbers. In the first simulation approaches that were developed, these random numbers were generated using manual techniques, such as flipping a coin and spinning a roulette (Rubinstein & Kroese, 2017). These methods were rapidly abandoned for at least three reasons: (1) the manual methods were far too slow for running advanced simulation models; (2) the generated sequences could not be reproduced; and (3) the generated sequences obtained were not always truly random. Nowadays, sequences of random numbers are generated using simple deterministic algorithms that can be easily implemented in computers. Hence, the corresponding generators are rather called ‘pseudo-random’ (Rubinstein & Kroese, 2017), and they are all built to produce sequences of numbers that are supposed to be uniformly distributed between 0.0 and 1.0.

Such pseudo-random algorithms must satisfy a certain number of properties for them to be ‘random enough’. They must pass a series of statistical tests demonstrating that they actually produce random numbers that are uniformly distributed in the $[0, 1]$ interval. Also, the random numbers generated must be independent, that is, a value in the generated sequence must not be related to the previous one(s). Since the corresponding computation is based on a sequence of numbers that use finite precision arithmetic, the sequence will repeat itself with a finite period, but this period must be as long as possible, and it must be much longer than the amount of random numbers needed for the simulation. Finally, as we will see below, these pseudo-random algorithms are computed from a starting seed, but both the randomness and the period must not depend on the initial seed used. Nowadays, most programming languages provide a build-in pseudo-random number generator, but not all of them accurately conform to the properties listed above, and they thus cannot all be used for scientific applications.

Just to make this a bit more concrete, the simplest methods that are used to generate pseudo-random sequences of numbers are called linear congruential generators, and were initially proposed by Lehmer (1951). Succinctly, they are based on the following recursive formula:

$$X_{t+1} = aX_t + c \pmod{m}\,.$$

The initial value, X_0, is called the seed, and a, c, and m are positive, integer constants. The expression ‘mod m’ means that, at each step t, the expression $aX_t + c$ is divided by m and the remainder is the generated value used for the next step. Careful choices for a, c, and m produce sequences of pseudo-random numbers that pass most of the statistical tests for uniformity, randomness, and independence mentioned above. A good example is the pseudo-random generator described by Lewis et al. (1969) with $a = 7^4$, $c = 0$, and $m = 2^{31} - 1$. However, such linear congruential generators are not frequently used anymore since they usually no longer totally meet the requirements of Monte Carlo simulation applications (see L'Ecuyer & Simard, 2007). Other methods are used, but their general frameworks are still based on linear recurrences like in the methods described above. The most successful and widespread one is called MRG32k3a and has been proposed by L'Ecuyer (1999). This is the pseudo-random generator currently implemented in most scientific computing languages, for example, in the R statistical and programming language (R Core Team, 2020). An implementation of this pseudo-random generator in C can be easily found online.

The tricky point regarding the use of these pseudo-random generators is to find a seed to start a Monte Carlo simulation. In very rare cases, the seed should remain the same between runs, and the generator will then produce the same sequence of random numbers. For example, this will be useful when there is a need to correct mistakes in a computer programming code being developed. However, in the vast majority of the cases, the seed must be different for each run of the simulation in order to generate independent replicates. In many applications, the system time of the computer in which the simulations are launched is used as a possible seed. For example, on computers running under the Linux operating system, such system time is expressed as the number of milliseconds that have elapsed since the 1st of January 1970 at 00h00m00s. Hence, since different runs of the same simulation will usually not start during the same millisecond, the seeds used will be different. Such a procedure, however, poses a problem since Monte Carlo simulations are now becoming more and more based on intensive computations that require long computation times. In such cases, the simulation runs are now frequently distributed on local or even worldwide computer grids. These are widely distributed, interconnected computers used to reach a common goal. In this case, there is a higher chance that several replicates will start at exactly the same time, and the corresponding replicates consequently will not produce different and independent results. Another method must thus be used to define the seed in each run of a simulation. On all computers running under the Linux environment, there is a special file that can be used for this. Its name is '/dev/urandom' and it gathers the environmental noise, for example, coming from device drivers. This noise can be read directly to generate possible seeds to launch the pseudo-random generator in each specific simulation run. Several works propose more advanced methods to set up the seeds in this case (see, e.g. Maigne et al., 2004).

3.4 Parameters and Their Statistical Distribution

Upon designing a Monte Carlo simulation model, two sets of parameters are usually considered. As seen above, state parameters are those on which simulations are based, and are the ones we vary when the outputs of the model are analysed (see the next section). The other parameters are sometimes called 'forcing parameters' and are used to define external influences acting upon the simulation process being addressed. In the example presented above, the state parameters of interest are the linear speed and the angular speed of the walking simulated females foraging for hosts to attack, while the forcing parameters are the total number of hosts and their spatial distribution in the environment.

State parameters have their own statistical distribution from which their values must be drawn, since Monte Carlo simulations are based on stochastic processes that are repeated multiple times. Such simulation processes are thus based on a statistical framework, and these methods are sometimes used to statistically estimate parameters. This is what is done, for example, when bootstrap techniques are used to estimate the robustness of statistically estimated phylogenetic trees (e.g. Augusta de Moraes & Selvatti 2018).

In this section, we will see how pseudo-random generators (that are producing random numbers uniformly distributed in the $[0, 1]$ interval, as we have seen above) can be used to draw sequences of random numbers from any kind of statistical distribution. There are different methods that can be used for such a purpose. The simplest one is called the inverse-transform method. Let us say we want to draw a value x from a statistical distribution having a cumulative distribution function $F(x)$. By definition, $F(x)$ is the probability that a variable X takes a value less than or equal to x. Such cumulative distributions always represent monotonically increasing functions from 0.0 to 1.0, as seen in two examples shown Fig. 3.5. These functions can be derived based on theoretically known distributions or they can be purely empirical. Once we have such a cumulative distribution function $F(x)$, the inverse-transform method consists of drawing a random value U uniformly from the $[0, 1]$ interval with a pseudo-random generator, and to deliver the value X that corresponds to the inverse function $F^{-1}(U)$, written as:

$$F^{-1}(U) = \inf\{X : F(x) \geq U\}\,.$$

This equation means that we are looking for the minimum value of x that satifies the equation $F(x) \geq U$. Figure 3.5 shows two examples of this procedure, one with a semi-quantative trait and another with a quantitative, continuously distributed trait.

A simple application of the inverse-transform method can be seen when we want to generate a random variable from a Bernoulli distribution. This distribution is a discrete distribution that takes the value 1 with probability p, and the value of 0 with probability $q = 1 - p$. This is frequently used in Monte Carlo simulation models when dealing with binary variables, for example to randomly draw the sex of an individual (male or female), whether a host is found or not by a parasitoid female, whether a host can escape or not from a parasitoid attack, etc. The cumulative distribution function in this case is similar to the one shown in the left panel of Fig. 3.5, but with only two possible outcomes. Therefore, the procedure to draw a random number having a probability p to appear (e.g. to decide if a host is attacked on not) is simply the following: first generate a random value U uniformly distributed in the $[0, 1]$ interval. Then, if $U \leq p$ return 1, otherwise return 0.

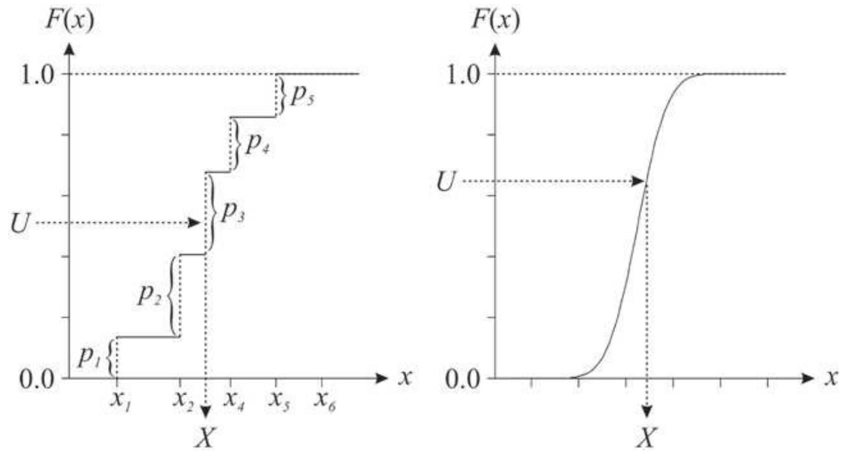

Fig. 3.5 Two examples of the inverse-transform method, with a semi-quantitative trait (left panel) or a continuously distributed trait (right panel), that can be used to draw a random number from any kind of statistical distribution. In both cases, the cumulative distribution function $F(x)$ is presented. A value U is drawn uniformly from the [0, 1] interval, and the delivered value X is given by the inverse function $F^{-1}(U)$. In the left panel, the p_i values are the frequencies in which each value x_i is or can be observed. Please note that the left panel could also represent the distribution of a purely qualitative trait. In this case, the method will work a similar way

The inverse-transform method can be applied for any kind of statistical distribution. For some of them, which are theoretically well known, more advanced (i.e. more efficient) methods are sometimes used instead. Boxes 3.1, 3.2, and 3.3 give the usual algorithmic procedures used to draw random numbers from an Exponential, a Normal, or a Poisson distribution, respectively.

Box 3.1
Drawing random numbers from an Exponential distribution.

Exponential distributions describe the distribution of times between events. In Monte Carlo simulations, such a distribution can be used, for example, to randomly draw the time elapsed between two host attacks by a parasitoid female. Exponential distributions are defined by only one parameter, the so-called rate parameter λ, and the expected value (i.e. the mean of the distribution) is known to be λ^{-1}. To draw a random number in an Exponential distribution having a rate parameter λ, the following algorithmic procedure can be used:

- Choose a parameter λ ($\lambda > 0$);
- Randomly draw a value U from a Uniform distribution between 0 and 1;

(continued)

Box 3.1 (continued)

- Compute $X = -\lambda^{-1} \ln(U)$;
- Return X.

If we want to draw a random number in an Exponential distribution having a mean μ instead, we just have to replace λ^{-1} by μ in the procedure above.

Box 3.2

Drawing random numbers from a Normal distribution.

Normal distributions are very common continuous distributions that can be used to describe many quantitative traits. In Monte Carlo simulations, they can be used, for example, to randomly draw traits such as length, size, weight, or distance. There are several methods that can be used to generate normally distributed random values. The most well known one is the so-called Box-Muller method that allows one to simultaneously draw two random, non-correlated values from a Normal distribution having a mean of 0 and a variance of 1. This works according to the following algorithmic procedure:

- Randomly draw two independent values U_1 and U_2 from a Uniform distribution between 0 and 1;
- Compute and return two random numbers X_1 and X_2 using the following equations:

$$\begin{cases} X_1 = \cos(2\pi U_2)\sqrt{-2\log(U_1)} \\ X_2 = \sin(2\pi U_2)\sqrt{-2\log(U_1)} \end{cases}.$$

If we want to draw numbers from a Normal distribution having a mean μ and a variance σ^2 instead, the obtained values must be additionally transformed as $\mu + X_1\sigma$ and $\mu + X_2\sigma$, respectively.

Box 3.3

Drawing random numbers from a Poisson distribution.

Poisson distributions are used to describe the discrete distribution of the number of events appearing during a given time interval. In Monte Carlo simulations, they can be used, for example, to randomly draw the number of eggs laid by a parasitoid female, the number of females a male can mate, etc. Poisson distributions are based on a single parameter $\lambda > 0$ that corresponds

(continued)

Box 3.3 (continued)
to the mean of the distribution. The following algorithmic procedure can be used to draw a random number from a Poisson distribution with mean λ:

```
• Choose $\lambda$ ($\lambda > 0$);
• Let $n = 0$ and $a = 1$;
• While $a \geq e^{-\lambda}$ do:
      Draw a value $U$ from a Uniform distribution between 0 and 1
      $a = aU$
      $n = n + 1$
• Compute $X = n - 1$;
• Return $X$.
```

Other algorithmic procedures are available to draw random numbers from any kind of distribution. There are even procedures that are available to draw random vectors of values from multidimensional distributions when several (e.g. correlated) values are needed in a Monte Carlo simulation. The interested readers should consult available textbooks, for example, Rubinstein and Kroese (2017).

3.5 Analysing the Obtained Results

Once the results of all simulations are collected, there is the need to analyse them both to check whether the computations produced suspicious outputs (pointing to some mistakes in the computer programming code) and to understand the messages delivered. For this, the results of each simulation run (or only averages and variances) are first saved. Then standard statistical descriptive tools can be used, such as summary statistics (means, ranges, variances, correlations, etc.). In this respect, graphic outputs are always needed, such as histograms, or simple plots showing the effects of different values of the forcing parameters on the mean and variance of the state parameters. Figure 3.3 gives an example of such a graph.

Another related descriptive tool that can be used is termed ‘sensitivity analysis’. The goal is to quantify the sensitivity of the model to changes in specific parameters. For this, the model is re-run fixing all parameters but the one we are interested in. For that one, different values are used sequentially, and changes in the corresponding average outputs are analysed to quantify the importance of this parameter on the model’s outputs. Repeating such a procedure for all parameters of the model can lead to a better understanding of the scientific question for which the Monte Carlo model has been designed.

Once all results are collected and described, it is tempting to perform statistical analyses to test the effect of, for example, a variation in one or several forcing parameters on the values of the state parameters of the model. Several authors take this approach, which is also sometimes requested by referees or editors of international journals. Although this is a continuously debated question, it is not valid to perform statistical comparisons on simulated data sets. Statistical tests are designed to be used when we do not know the real values of the parameters we are estimating, and parameter estimation should be done by sampling the populations. In this case, we need a statistical procedure (usually based on the parameters' distributions) to compare them. In a simulation work, however, parameters such as means are explicitly known. We only have to run a sufficient amount of simulations to know them exactly. There is thus no distribution (since parameters are not estimated from some sampling procedures), and thus statistical comparison procedures are not needed. In other words, by running simulations several times, we always end up with a statistically significant test, since the standard errors will mechanically tend to zero. The only thing that can be done is to present effects, with the descriptive statistical tools mentioned above, discussing trends, etc. (see, e.g. White et al., 2014).

3.6 Looking for Optimised Values

For some scientific questions, we are willing to address using Monte Carlo simulations, and once the simulations have been run and the results obtained and analysed, we can go one step forward. We sometimes need to know what values of the state parameters maximise some pre-defined objective criterion. In behavioural ecology, for example, we would often like to identify the set of state parameters that define the behaviour of the simulated animals maximising the overall number of progeny produced. In defining an efficient biological control programme against a crop pest, we might alternatively be interested in identifying the value of the parameters that maximise the number of hosts killed per time unit or the crop production, etc.

There are different numerical ways to achieve this goal. One popular method is the use of genetic algorithms. This approach is efficient, simple, and relatively easy to implement. The method was invented in the early 1970s to mimic the process of natural evolution (Holland, 1975; Goldberg, 1989) and has been used since then to solve several problems in the field of behavioural ecology and evolutionary biology (see, e.g. Sumida et al., 1990; Mitchell, 1998; Hoffmeister & Wajnberg, 2008; Ruxton & Beauchamp, 2008; Wajnberg et al., 2012; Hamblin, 2013; Wajnberg et al., 2013; Plouvier & Wajnberg, 2018).

The main processes involved in a genetic algorithm are shown in Fig. 3.6. At the beginning, a set (or 'population') of possible solutions are drawn randomly,

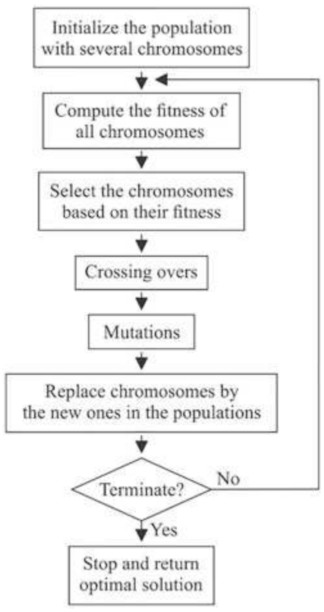
Initialize the population with several chromosomes
Compute the fitness of all chromosomes
Select the chromosomes based on their fitness
Crossing overs
Mutations
Replace chromosomes by the new ones in the populations
Terminate?
No
Yes
Stop and return optimal solution

Fig. 3.6 Flowchart showing the essential components of a genetic algorithm (see text for explanations)

using the procedures described in the sections above. The word ‘solution’ actually means a list of values for all state parameters (that are called here ‘genes’) of the Monte Carlo simulation problem. These are arranged sequentially along a so-called chromosome. Running the simulation model on each of the initial chromosomes leads to an estimate of their ‘fitness’ through the pre-defined objective function of the model. Then, pairs of ‘parents’ for the next generation are selected randomly in the initial population of chromosomes using a probability proportional to their fitness. Hence, chromosomes with higher fitness have a higher chance to contribute to the next generation. There are several methods to implement this. The interested reader can consult Hoffmeister and Wajnberg (2008) or Hamblin (2013) for additional explanations. Once the pairs of parents are identified, they may undergo crossing over events, which means that recombination is performed, like on real chromosomes in biology (Hoffmeister & Wajnberg, 2008; Hamblin, 2013). Finally, each parameter (gene) of the new children obtained can pass through a mutation process that usually implies adding to their value an adjustment drawn from a Normal distribution having a mean of 0.0. Then, the new chromosomes (children) are used to build a new generation and the process is repeated until a stopping criterion is reached. Here again, several stopping criteria can be used. For example, the looping process can be stopped when a fixed number of generations have been completed (e.g. Barta et al., 1997; Wajnberg et al., 2013) or after a fixed computation time. Usually, however, the performance of the genetic algorithm in producing an optimal solution is followed over the course of multiple generations, and the process is then stopped if a fixed number of generations has passed without a substantial improvement in the fitness of the best chromosome.

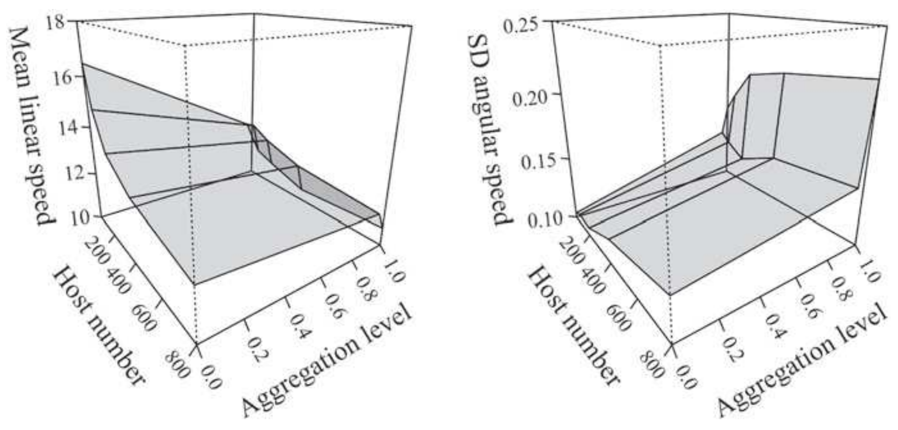

Fig. 3.7 Optimised (with the use of a genetic algorithm) mean linear walking speed (expressed in cell unit; left panel) and SD of the angular speed (expressed in radian; right panel) maximising the fitness (i.e. number of hosts discovered and attacked per time unit) of simulated parasitoid females foraging in a 2D space for hosts present in different density, and having different spatially aggregate levels. The host aggregation level ranges from 0.0, indicating that hosts are randomly distributed, to 1.0, in which all hosts are clumped into a single patch. Increasing values of this parameter correspond to a decreasing number of host patches in the environment

Such an optimisation procedure has been used for the simple didactic example presented in the first section of this chapter (simulating the walking behaviour of a single parasitoid female foraging for hosts having an aggregate distribution in a 2D space), and the optimal solution found is shown in Fig. 3.3. The same Monte Carlo simulation model was then run with different host densities, and with different host aggregation patterns in the environment, and a genetic algorithm was used in each situation to find the optimised linear walking speed and SD of the angular speed, which describes the tendency of the animal to walk on a straight line or not. The results obtained are presented in Fig. 3.7. They indicate that optimised parasitoid females foraging for hosts in a 2D space should walk slower and less on a straight line when there are most hosts available in the environment. When hosts become more aggregated, optimised females should also walk slower and less on a straight line.

3.7 A More Advanced Example

Plouvier and Wajnberg (2018) developed a Monte Carlo simulation coupled with the use of a genetic algorithm to find what biological and ecological features of natural enemies should optimise their pest control efficacy on a field crop. The economic income collected by the farmer when releasing biological control agents was also considered. For this, the model was designed to optimise a criterion corresponding to the total crop yield (expressed as plant biomass), taking into account damage done by the pest, minus the overall cost of producing and releasing natural enemies.

The simulations were done using a spatially explicit framework simulating a field represented by a grid in which crop plants are grown in rows, one every second row. The state parameters optimised represented eight life-history and behavioural traits of the released natural enemies: (1) longevity (expressed as time steps before dying), (2) fecundity (expressed as the total number of eggs corresponding to the total number of hosts killed if the natural enemy is a solitary parasitoid), (3) pest attack rate, per time unit, (4) level of local intra-specific competition (i.e. interference) leading to a reduction in the pest attack rate, (5) pest handling time (expressed in time steps), (6) probability to move/disperse, and (7) the mean and (8) SD of the (Normal) distribution of the distance moved (both expressed in grid cell unit). Five forcing variables were analysed: (1) the number of sites in the field in which the natural enemies are released (1 vs. 2), (2) the number of individuals released (10 vs. 20), (3) the timing of natural enemies release (6 vs. 10 time steps after the crop has been sown), (4) the cost of producing and releasing the natural enemy (200 vs. 300 arbitrary units), and (5) the growth rate of the plant ($r = 0.05$ vs. $r = 0.06$). Finally, several other features were added to the model. For example, a negative trade-off between longevity and fecundity of the natural enemies was considered, as this is observed in real situations (see, e.g. Miyatake, 1997). The plant development has been modelled with an exponential growth function with a decreasing rate. Pests arrive on the crop at time step 5 (after the crop has been sown) and are distributed randomly over the entire crop. Natural enemies are attacking pests following a type II functional response (Holling, 1959), and they can disperse at each time step with a probability that increases with the local density of competitors, but decreases with the local density of pests.

Briefly, the results obtained demonstrated that ideal natural enemies, that is, ones that optimise the income collected by the farmer using a biological control approach, should have a shorter longevity, and hence a higher fecundity. They should also have a lower pest handling time leading to increase the overall number of pests attacked on each plant. Finally, they should have a higher tendency to disperse when the local pest density is low. On the other hand, pest attack rate and the level of competition (i.e. interference) between natural enemies surprisingly appear to be less important.

Results obtained also indicated that the more important ecological and behavioural features defining efficient natural enemies are correlated, demonstrating a kind of an optimised ‘behavioural syndrome’ (Sih et al., 2004a, b). Also, most of the optimised parameters actually had a bimodal distribution, indicating that there were actually two optimised pest control strategies. The first one, called ‘incremental strategy’, corresponds to low pest handling time, associated to a low tendency to disperse. In this case, the natural enemies are ‘cleaning’ the plants from pests before going to another plant. The second one, called ‘decremental strategy’, on the contrary, is associated to a high pest handling time with a high dispersion tendency, corresponding to a natural enemy that attacks fewer hosts in a local area before leaving but covers a larger crop surface area. The ability of each of these two pest control strategies to perform well depends on the number of natural enemies released and on the number of release points (Plouvier & Wajnberg, 2018).

3.8 Conclusion

Monte Carlo simulations have been demonstrated to be a very efficient modelling tool that can be used when the problems to be addressed are too complicated (e.g. based on too many parameters) to be handled with standard, analytical modelling approaches. In this respect, this chapter gives a general overview on how such a modelling framework can be implemented, for example using a programming language like C or C++, with minimum skill. There are currently some other computer languages that might sometimes be more convenient to use for developing such simulations with minimal programming effort. For example, in the R statistical and programming environment (R Core Team, 2020), there are built-in functions enabling users to draw random numbers from all basic statistical distributions, hence without having to code this explicitly. However, using such computer languages usually produce simulation models that need more computer time to run, and that are not easily distributed on several computers simultaneously.

In this respect, another point that must be mentioned here is that computers are still regularly becoming more and more powerful and able to provide faster processing times. Moreover, nowadays computers are interconnected through networks in local or even worldwide computer grids offering powerful computing resources providing a way to distribute simultaneously long-lasting simulations leading to delivery of results in reasonable times. Such distributed computer resources enable researchers to address highly complex scientific problems that are especially, but not only, developed to understand the ecology of systems involving agricultural pests and their natural enemies.

Acknowledgements All computations done to prepare this chapter were achieved using the biomed virtual organisation of the EGI infrastructure, with the dedicated support of resource centres BEINJING-LCG2, IN2P3-IRES, OBSPM, INFN-FERRARA, GRIF, INFN-CATANIA, INFN-ROMA3, INFN-BARI, CREATIS-INSA-LYON, NCG-INGRID-PT, INFN-PISA, CESNET-MCC, and CLOUFIN, resource centres in UK hosted by GridPP collaboration, and the additional support of the resource centres listed here: http://operations-portal.egi.eu/vapor/resources/GL2Browser?VOfilter=biomed. I also thank Andrei Tsaregorodtsev for his help in using this computer grid, and George Heimpel for his comments on a previous version of the manuscript. Finally, this work was supported by the Israel Institute for Advanced Studies, in Jerusalem, in which a major part of this chapter was written.

References

Augusta de Moraes, R. C., & Selvatti, A. P. (2018). Bootstrap and rogue identification tests for phylogenetic analyses. *Molecular Biology and Evolution, 35*, 2327–2333.

Barta, Z., Flynn, R., & Giraldeau, L. A. (1997). Geometry for a selfish foraging group: A genetic algorithm approach. *Proceedings of the Royal Society of London Series B Biological Science, 264*, 1233–1238.

Bruins, E. B. A. W., Wajnberg, E., & Pak, G. A. (1994). Genetic variability in the reactive distance in *Trichogramma brassicae* after automatic tracking of the walking path. *Entomologia Experimentalis et Applicata, 72*, 297–303.

Clark, C. W., & Mangel, M. (2000). *Dynamic state variable models in ecology – Methods and applications*. Oxford University Press.

Giró, A., Padró, J. A., Valls, J., & Wagensberg, J. (1985). Monte Carlo simulation of an ecosystem: A matching between two levels of observation. *Bulletin of Mathematical Biology, 47*, 111–122.

Giró, A., Valls, J., Padró, J. A., & Wagensberg, J. (1986). Monte Carlo simulation program for ecosystems. *Computer Applications in the Biosciences (Cabios), 2*, 291–296.

Godfray, H. C. J. (1994). *Parasitoids. Behavioral and evolutionary ecology*. Princeton University Press.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, Longman Publishing, Inc.

Hamblin, S. (2013). On the practical usage of genetic algorithms in ecology and evolution. *Methods in Ecology and Evolution, 4*, 184–194.

Hoffmeister, T. S., & Wajnberg, E. (2008). Finding optimal behaviors with genetic algorithms. In E. Wajnberg, C. Bernstein, & J. van Alphen (Eds.), *Behavioral ecology of insect parasitoids – From theoretical approaches to field applications* (pp. 384–401). Blackwell Publishing.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. The University of Michigan Press.

Holling, C. S. (1959). Some characteristics of simple types of predation and parasitism. *The Canadian Entomologist, 91*, 385–398.

Houston, A. I., & McNamara, J. M. (1999). *Models of adaptive behaviour – An approach based on state*. Cambridge University Press.

L'Ecuyer, P. (1999). Good parameters and implementations for combined multiple recursive random number generators. *Operations Research, 47*, 159–164.

L'Ecuyer, P., & Simard, R. (2007). TestU01: A C library for empirical testing of random number generators. *ACM Transactions on Mathematical Software, 33*, 22.

Lehmer, D. H. (1951). Mathematical methods in large-scale computing units. *Annals of the Computation Laboratory of Harvard University, 26*, 141–146.

Lewis, P. A., Goodman, A. S., & Miller, J. M. (1969). A pseudo-random number generator for the system/360. *IBM Systems Journal, 8*, 136–146.

Maigne, L., Hill, D., Calvat, P., Breton, V., Reuillon, R., Lazaro, D., Legre, Y., & Donnarieix, D. (2004). Parallelization of Monte Carlo simulations and submission to a grid environment. *Parallel Processing Letters, 14*, 177–196.

Maynard-Smith, J. (1982). *Evolution and the theory of games*. Cambridge University Press.

Metropolis, N. (1987). The beginning of the Monte Carlo method. *Los Alamos Science. Special Issue, 1987*, 125–130.

Mitchell, M. (1998). *An introduction to genetic algorithms*. MIT Press.

Miyatake, T. (1997). Genetic trade-off between early fecundity and longevity in *Bactrocera cucurbitae* (Diptera: Tephritidae). *Heredity, 78*, 93–100.

Plouvier, N. W., & Wajnberg, E. (2018). Improving the efficiency of augmentative biological control with arthropod natural enemies: A modeling approach. *Biological Control, 125*, 121–130.

R Core Team. (2020). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing. https://www.r-project.org/

Roitberg, B. D. (1985). Search dynamics in fruit-parasitic insects. *Journal of Insect Physiology, 31*, 865–872.

Rubinstein, R. Y., & Kroese, D. P. (2017). *Simulation and the Monte Carlo method* (3rd ed.). Wiley.

Ruxton, G. D., & Beauchamp, G. (2008). The application of genetic algorithms in behavioural ecology, illustrated with a model of anti-predator vigilance. *Journal of Theoretical Biology, 250*, 435–448.

Sih, A., Bell, A., & Johnson, J. C. (2004a). Behavioral syndromes: An ecological and evolutionary overview. *Trends in Ecology & Evolution, 19*, 372–378.

Sih, A., Bell, A. M., Johnson, J. C., & Ziemba, R. E. (2004b). Behavioral syndromes: An integrative overview. *The Quarterly Review of Biology, 79*, 241–277.

Sumida, B. H., Houston, A. I., McNamara, J. M., & Hamilton, W. D. (1990). Genetic algorithms and evolution. *Journal of Theoretical Biology, 147*, 59–84.

Wajnberg, E., Bernstein, C., & van Alphen, J. (2008). *Behavioral ecology of insect parasitoids – From theoretical approaches to field applications*. Blackwell Publishing.

Wajnberg, E., Coquillard, P., Vet, L. E. M., & Hoffmeister, T. (2012). Optimal resource allocation to survival and reproduction in parasitic wasps foraging in fragmented habitats. *PLoS ONE, 7*(6), e38227.

Wajnberg, E., Hoffmeister, T. S., & Coquillard, P. (2013). Optimal within-patch movement strategies for optimising patch residence time: An agent-based modelling approach. *Behavioral Ecology and Sociobiology, 67*, 2053–2063.

White, J. W., Rassweiler, A., Samhouri, J. F., Stier, A. C., & White, C. (2014). Ecologists should not use statistical significance tests to interpret simulation model results. *Oikos, 123*, 385–388.