Emulation of natural selection through a genetic algorithm in a prey-predator model

Lorenzo Pantolini

May 14, 2018

Contents

Chapter 1

Introduction

The aim of the work is to reproduce the evolution process through an *agent-based* simulation. We built a *prey-predator* model in which preys are characterized by parameters that determine their survival in the simulated environment. Preys feed themselves with resources provided by the environment and try to escape from predators. They reproduce generating individuals whose parameters are a combination of those of parents. *Genetic algorithms* regulate the entire reproductive process by fostering the reproduction of the most suitable individuals. Due to this model we expect that the parameters of next generations will converge to the best choice for the survival.

We wrote the simulation in NetLogo, while the routine of the genetic algorithm is in Python.

1.1 Environment

Agents live in a two-dimensional grid divided into small squares called "patches". Some of these are "fertile" which means that resources will grow there periodically. Before the simulation starts it is possible to set some parameters in order to define the abundance of the habitat:

- *fertility*: the percent of fertile patches.
- *grass-regrowth-time*: the time it takes to re-grow resources.
- *environment-prosperity*: the amount of resources growing per epoch.
- *max-grass*: the maximum value of resources that a patch can contain.

The possibility to change the environment allows us to observe how agent parameters change as a function of the amount of available resources.

1.2 Predators

Predators are simple agents, we can almost see them as a feature of the environment. They do not die neither reproduce. Their only behavior is to move randomly until they find a prey to attack. Once they found it, their chance of killing it is a probability that depends on the sizes of both the agents. Two parameters characterize them: *Vision* and *Dimension*, which are common to all predators. From the main interface it is possible to set:

- *predators-number.*
- *predators-vision.*
- *predators-dimension.*

Again the freedom on parameters allows us to observe the different evolution of preys according to the characteristics of the predators.

1.3 Preys

Preys are far more complex than predators. These follow all the same behavioral rules whose efficiency depends on five parameters that characterize each agent. We can see these parameters as a sort of "genotype" that determines, for each individual, the ability to survive in the environment. It is exactly this genotype that is manipulated by the genetic algorithm and transmitted to next generations.

1.3.1 Behavioral rules

Preys feed themselves with resources provided from the environment and at the same time worry about surviving the predators. At each step of the simulation they observe if there are predators in their visual range, if there is:

- *one predator:* the prey "escapes", which means that it moves in the patch in the opposite direction from the predator.
- *more than one predator:* the prey "escapes" from the predator it identifies first.[1]
- *no predator:* the prey evaluates if it needs feeding (meaning its energy is not maximal) and if the patch where it is located contains resources:

[1] it is not a particularly efficient escape technique, we will discuss later the implications of this choice.

- *if so:* it feeds itself.
- *if not:* it moves to the patch with more resources between the 8 adjacent ones.

The code that produces this routine is reported in A. Preys have also an internal *energy* which is used to move, if they run out of that stamina they die. They can recover energy by converting resources provided from the environment during the feeding process. However, stored energy has a maximum value, function of the "genetic parameters" of the individual. We explain all the details in the next section after describing the genetic parameters.

1.3.2 Genetic parameters

The five parameters that characterize each prey are: *Vision, Dimension, Voracity, Capacity* and *Pleasantness*. Each of them has advantages and disadvantages, so that the particular combination that guarantees the best result in terms of survival is not obvious.

Vision:
vision is the range in which preys can spot a predator. This allows them to escape before they are spotted and then attacked. We observe that a too high vision keeps preys in constant alarm and therefore always on the run, leaving no time to feed and recover energy.

Voracity:
voracity is the maximum amount of resources converted into energy during the feeding process (the energy-resources ratio is 1:1). The absorbed resources are equal to the maximum value between the voracity and the resources present in the patch in which the agent is located. Furthermore, this parameter influences the maximum energy that can be stored by the individual, we will see the precise relationship later.

Capacity:
also this parameter contributes to the ability to store energy, however carrying a big amount of resources leads to a greater "energy cost" during the movement.

Dimension:
dimension is an alternative to escape when preys are attacked by predators, in fact they have a chance of survival equal to:

$$P_{survival} = \frac{prey\ Dimension}{predator\ Dimension} \tag{1.1}$$

This could allow preys not to fear predators in case they exceed their size. However, imposing dimensions imply a greater energy expenditure during the movement, so focusing on size may not be the most appropriate evolutionary choice.

Pleasantness:
pleasantness is irrelevant for the fate of preys, it has been inserted only to observe the different evolution with respect to the other parameters.

1.3.3 Energy

Let us show the relationships between energy and genetic parameters:

Energy:
used to perform movements and reloaded by feeding. As we said, it is very important because if a prey runs out of energy it will die. So this parameter will play a crucial role in the selection of individuals destined for reproduction.

Maximum energy:
this, as the name suggests, is the maximum value of storable energy. When the energy is maximum ($E = E_{max}$) the agent will not try to feed itself:

$$E_{max} = \ Capacity \cdot Voracity + 1 \tag{1.2}$$

Energy cost:
this is the energy cost of a movement:

$$E_c = Dimension + Voracity + \frac{Capacity}{5} \tag{1.3}$$

1.3.4 Reproduction

At the simulation starts it is possible to set the number of preys with the label *preys-number*. Their genetic parameters are randomly generated, they are integers between 1 and 50. Preys reproduce themselves in couples, at each temporal step 2 of them are selected and their genome are recombined by a genetic algorithm. In the next chapter we explain the details of reproduction.

Chapter 2

Genetic algorithms

Genetic algorithms are inspired by the process of natural selection and belong to the larger class of evolutionary algorithms. They are commonly used to generate high-quality solutions to optimization and search problems by relying on bio-inspired operators such as **mutation, crossover** and **selection**. In a genetic algorithm, a population of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem is evolved toward better solutions. Each candidate solution has a set of properties (its chromosomes or genotype) which can be mutated and altered; traditionally, solutions are represented in binary as strings of $0s$ and $1s$. Every individual in the population has a *fitness* which usually is the value of the objective function in the optimization problem. The more fit individuals are stochastically selected from the current population, and each individual's genome is modified (recombined and possibly randomly mutated) to form a new generation. The new generation of candidate solutions is then used in the next iteration of the algorithm. Commonly the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population (see Genetic algorithms).

There are many ways to implement such algorithms, let's show our choices for *fitness*, *selection* and *recombination of "genome"*.

2.1 Fitness

In our work we want to find the individuals who survive better in the given environment. The survival of the preys is strongly linked to their energy since they die when it is lower than 0, so we have built the fitness as:

$$F(t) = \frac{\sum_{k=t_0}^{t} E_k}{t \cdot E_{max}} \tag{2.1}$$

where:

- t is the time in which the fitness is evaluated.
- t_0 is the instant in which the prey is born.
- E_k is the energy of the prey evaluated at the instant k.

In this way the preys with a big amount of energy over the time will have a good fitness.

2.2 Selection procedure

In the selection process at every prey is associated a probability to be chosen proportional to their fitness value (calculated in that instant). The input given to the algorithm is a list which contains the five genetic parameters of every prey and their fitness:

$$input = [prey_1, \ ... \ , prey_N]$$

$$prey_i = [Dimension_i, Vision_i, Voracity_i, Capacity_i, Pleasantness_i, Fitness_i]$$

the algorithm:

- extracts the fitness of all the preys.
- builds a segment of extremes 0 and $Max = \sum_{i=1}^{N} fitness_i$.
- picks a random number x_1 between $[0, Max]$.
- selects the sub-segment in which x_1 falls and chooses the corresponding prey.

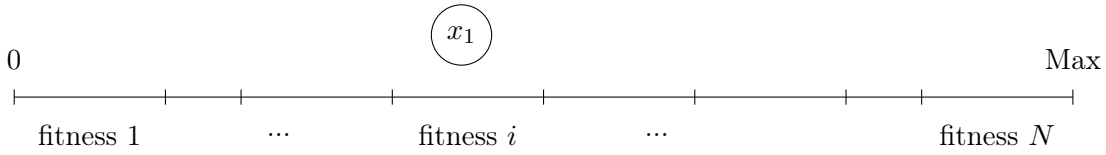

Figure 2.1: example of selection segment

The second prey is chosen in the same way, but in order to avoid the reproduction between an agent and itself the chosen sub-segment is removed after the first selection. Notice that this criterion of selection associates an higher probability to be chosen to the individuals with a higher fitness value.

2.3 Recombination of "genome"

Once obtained the two "reproducers" the algorithm proceeds to the recombination of their genetic parameters. The inputs are two lists containing the genome of the selected preys:

$$prey_1 = [Dimension_1, Vision_1, Voracity_1, Capacity_1, Pleasantness_1, Fitness_1]$$

$$prey_2 = [Dimension_2, Vision_2, Voracity_2, Capacity_2, Pleasantness_2, Fitness_2]$$

the algorithm:

- removes the fitness from the lists.
- converts the parameters in binary and puts them in two strings.
- picks a random number y between 0 an the strings length[1].
- "cuts" the strings in position y and all data beyond that point in either string is swapped between the two parent sequences.

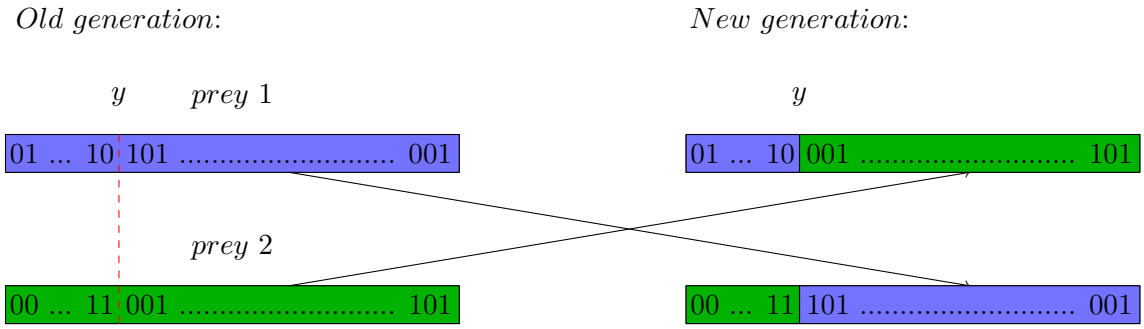

Figure 2.2: example of crossover: on the left the parents, on the right the new generation.

[1] which is the same since they have the same number of parameters.

In nature during the copy of the genotype an error may occur that leads to a mutation. In order to reproduce this operation every new prey has a "mutation probability" (that can be set in the simulation with the label *prob-mutation*). If mutation occurs a random digit of the string of genetic parameters is changed, if it is a 0 it becomes 1 and vice versa. The output of the algorithm is a list which contains the genetic parameters of the two resulting preys.

Chapter 3

Results

In this chapter we observe the mean value of the genetic parameters, the fitness and other values of interest.

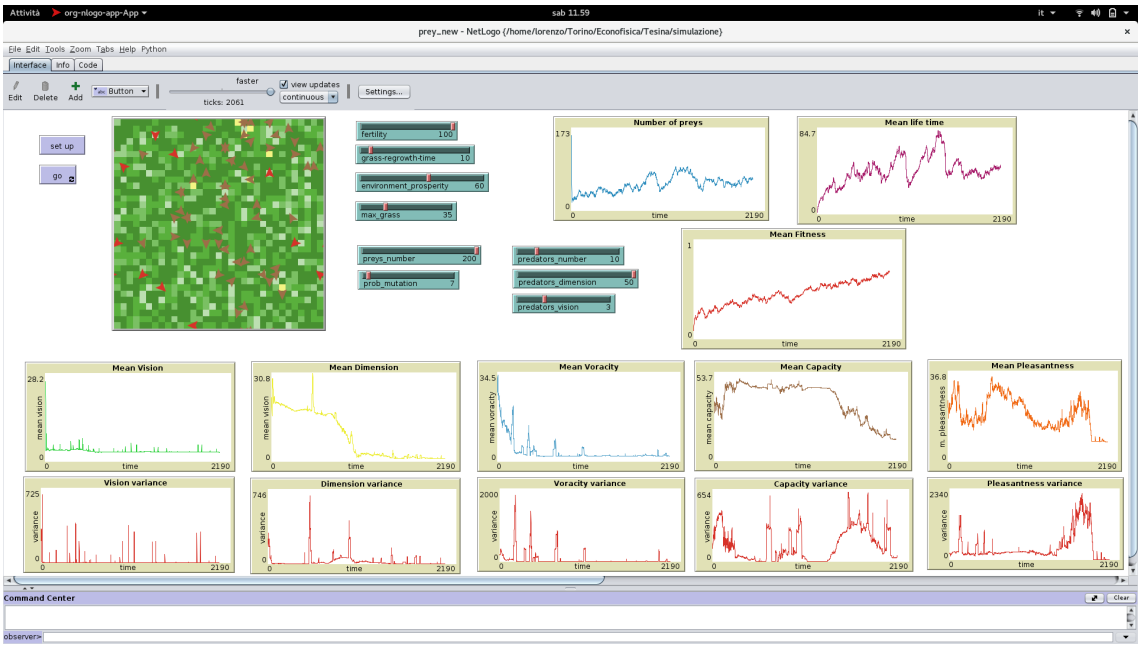

Figure 3.1: screenshot of of a run of the simulation.

The parameters used for this particular simulation run are:

- $fertility \ = \ 100$
- $grass \ regrowth \ time \ = 10$
- $environment \ prosperity \ = 60$

- $max\ grass\ = 35$
- $preys\ number\ = 200$
- $prob\ mutation\ = 7$
- $predators\ number\ = 10$
- $pedators\ dimension\ = 50$
- $predators\ vision\ = 3$

3.1 Quantities of interest

We observe that the population decreases really fast in the first part of the simulation. This is due to the genetic parameters of the first generation of preys. In fact they are randomly generated, so most of the preys are not suitable to survival. However, as we expected, after few generations the survival skills of the individuals improve. This successfully leads to the increment of the fitness, the population size and the average life time of the individuals.

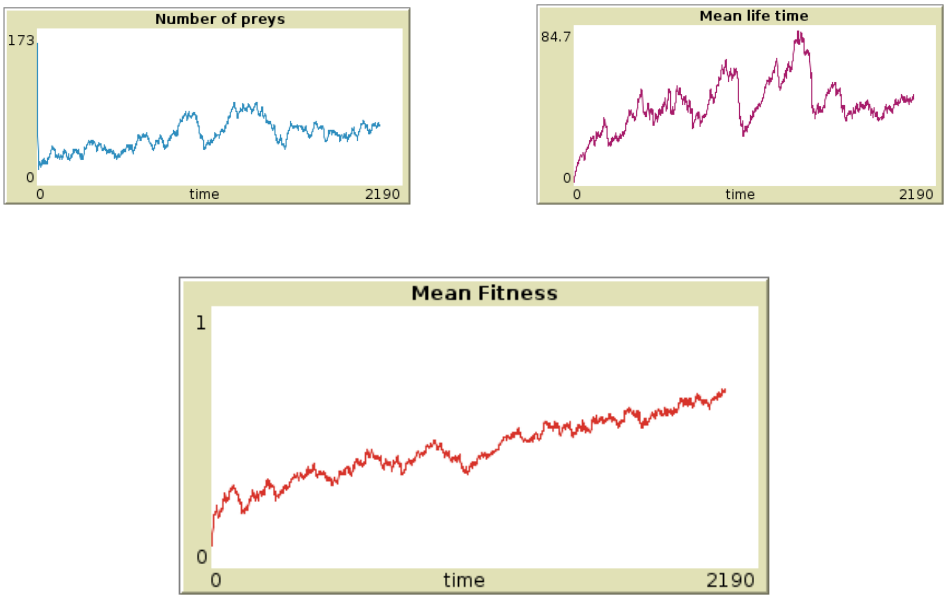

Figure 3.2: population size, average life time and average fitness in the first 2190 steps of the simulation.

3.2 Genetic parameters

Now we focus on the evolution of the mean values of the genetic parameters over time.

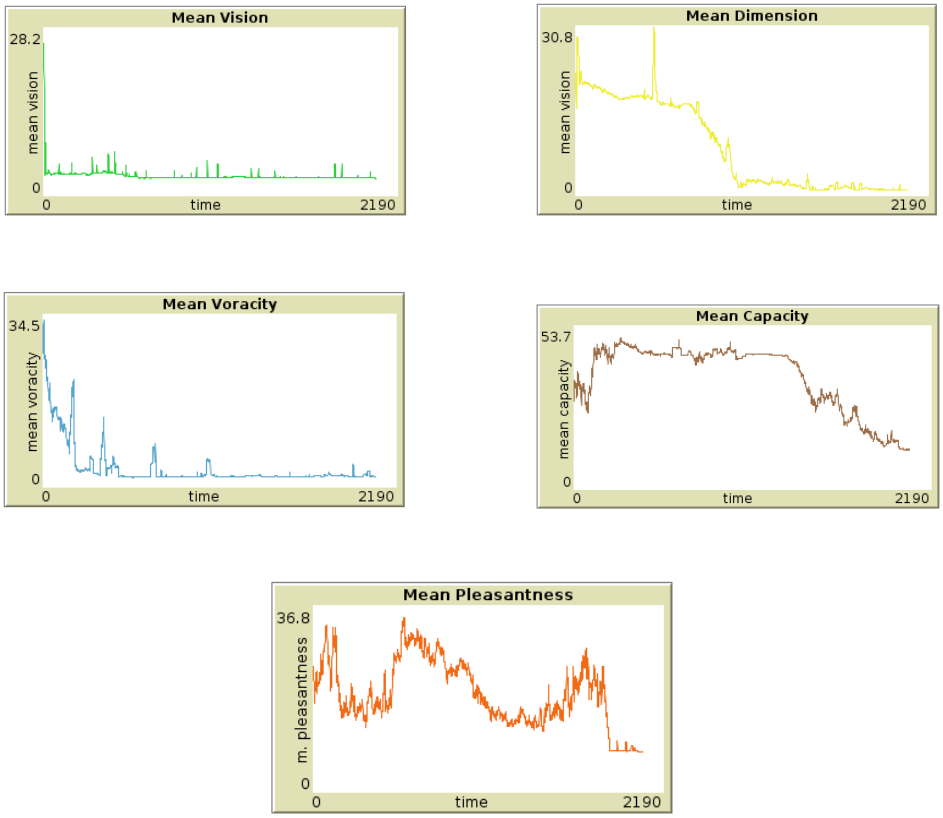

Figure 3.3: mean values of the genetic parameters.

As we can see these parameters converge in different times (except *pleasantness* which does not converge). The *vision* of preys is the fastest, it converges rapidly to 3 which is the exactly vision of the predators in this run. We could expect this result. In fact in the simulation preys act before predators, so with the same vision they are safe. The *dimension* goes to 0, from this we deduce that preys do not rely on it to survive (see equation (1.1)).[1] We also note that the *capacity* has a less regular trend than the other parameters (except *pleasantness*). It needs more time to converge as we will see in the next section. At last we have the *pleasantness*, the "free" parameter. As expected it has an irregular trend, but unlike the capacity it will never converge.

[1] at least with these specific predators. After we will see another kind of choice.

3.3 Final results

Letting the simulation to continue we observe the convergence of the *capacity*, while the *pleasantness* still does not converge.

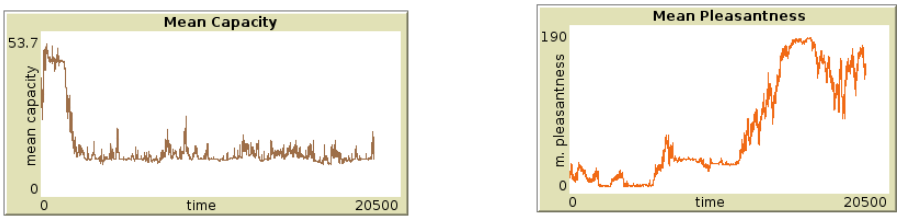

Figure 3.4: mean value of capacity and pleasantness in the first 20500 steps of the simulation.

Since all the relevant parameters have converged, reaching the optimal value, the fitness has stopped growing. Although agents have excellent parameters to survive in this environment, the population fluctuates around a constant value as resources are limited.

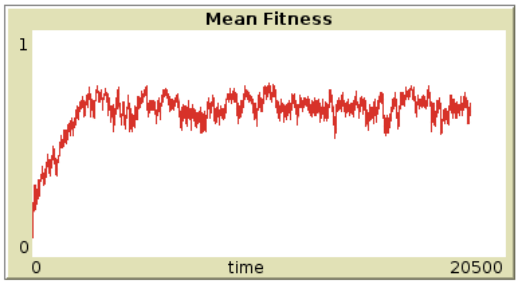

Figure 3.5: average fitness in the first 20500 steps of the simulation.

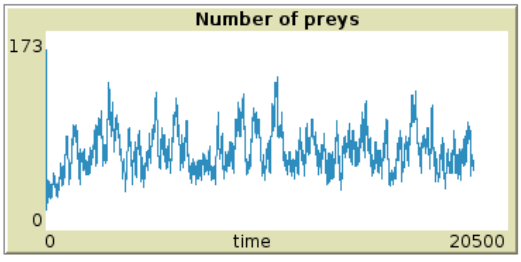

Figure 3.6: population size in the first 20500 steps of the simulation.

3.4 A different run

Now we observe the results obtained by increasing the number of predators and decreasing their size. We setup the same initial parameters of the previous simulation, except two differences:

$$predators\ number: \ 10 \rightarrow \ 15 \tag{3.1}$$

$$predators\ dimension: \ 50 \rightarrow \ 20 \tag{3.2}$$

Now we pay attention to the *vision* and the *dimension* of this new population.

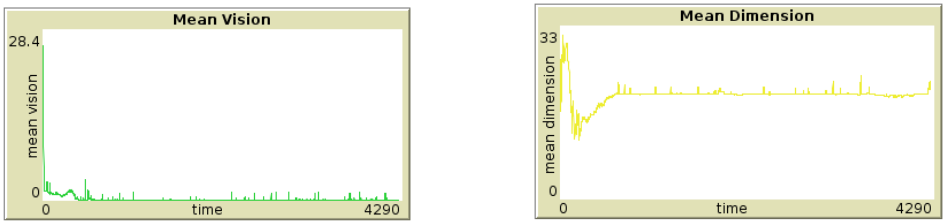

Figure 3.7: mean value of vision and dimension of the new population.

Also this time the parameters converge quickly, but now they assume a different value. *Dimension* converges to 20 which means that predators cannot kill preys (see equation (1.1)). Notice that a higher value would be useless in terms of survival and detrimental in terms of energy costs (see equation (1.3)). Since preys do not have to fear predators they do not even need to see them. In fact this time their *vision* converges to 0. This means that preys are blind, so they ignore predators. In this way they avoid wasting time fleeing.

This example shows perfectly how a different environment leads to a different "evolutionary choice".

3.5 Conclusions

As we have seen the genetic algorithm we implemented successfully leads to individuals suitable to survival in a specific environment. We observe that the model can be improved in several ways, for example we could use continuous genetic parameters instead of discrete ones. Furthermore, making the predators more complex and able to reproduce would be interesting. This would allow us to observe the process of *coevolution* between prey and predator.

Appendix A

Preys routine

In this appendix we present the behavioral routine of the preys.

```
to go

  ask preys [
              act
              set Fitness (Fitness + Energy / Max_energy)
              set Life_time Life_time + 1

              set Fitness_on_time (Fitness / Life_time)
            ]

    .
    .
    .

 end
```

Where "act" is the following function:

```
to act

  let n_enemies count predators in-radius Vision

  ifelse n_enemies > 0 [
                         let enemy one-of predators in-radius Vision
                         face enemy
                         set heading ( heading + 180 )
                         set Energy ( Energy - Energy_cost )
                         if Energy < 0 [ die ]
```

```
                        forward 1
                      ]

                      [
                         ifelse resource > 0 and Energy < Max_energy [ feed ]
                                                                     [ move ]
                      ]

end
```

while "feed" and "move" are:

```
to feed

  let max_meal Voracity      ;max amount of food he can eat

  ifelse resource >= max_meal [
                                set Energy ( Energy + max_meal )
                                set resource ( resource - max_meal )
                              ]

                              [
                                set Energy ( Energy + resource )
                                set resource 0
                              ]

  if Energy > Max_energy [ set Energy Max_energy ]


end



to move

  set Energy ( Energy -  Energy_cost )

  if Energy <= 0 [
                   die
                 ]

 move-to patch-here                        ; go to patch center
```

```
 let p max-one-of neighbors [resource]
 if [resource] of p >= resource [
                                  face p
                                  move-to p
                                 ]

end
```

Appendix B

Genetic algorithm

The genetic algorithm was written in Python. Here we append the code for the selection process.

```
from numpy import random

def selection ( DNAs ):

        DNA = DNAs[:]
        probability_bar = 0

        for dna in DNA:
                probability_bar += dna[-1]
                #the last element is the fitness



        x = probability_bar*random.random()
        winner1 = []
        lower_bound = 0
        j = 0

        #check in which interval x falls
        for dna in DNA:

                if( x <= dna[-1] + lower_bound ):
                        winner1 = dna
                        probability_bar -= dna[-1]  #update the segment
                        del DNA[j]
                        break
```

```
            lower_bound += dna[-1]  #update the lower bound
            j += 1

        #same routine for the second one
        y = probability_bar*random.random()
        winner2 = []
        lower_bound = 0
        j = 0

        for dna in DNA:

            if( y <= dna[-1] + lower_bound ):
                winner2 = dna
                break

            lower_bound += dna[-1]
            j += 1


        winners = [ winner1, winner2 ]

        return winners
```

Now we show the code which manipulates the "genotypes" of the selected preys.

```
import selection as sl
from numpy import random

#mutation routine
def error_prbability( dna, prob_err ):

    if ( random.randint(1,100) <= prob_err ):

        n_error = random.randint( 0,(len(dna)-1) )

        if ( dna[n_error] == '0' ):
            dna = dna[:n_error] + '1' + dna[n_error+1:]

        elif ( dna[n_error] == '1' ):
            dna = dna[:n_error] + '0' + dna[n_error+1:]
```

```
        return dna


def genetic_algorithm( DNA, prob_err ):

        winners = sl.selection(DNA)
        dna1 = ''
        dna2 = ''


        for x in winners[0][:-1]:
                #converts the elements in binary and put them in a string
                dna1 += bin(x)[2:].zfill(8)


        for y in winners[1][:-1]:

                dna2 += bin(y)[2:].zfill(8)


        N_max = len(dna1) - 1     #maximum cut

        cut = random.randint(1,N_max)

        #define the output
        dna1_f = dna1[:cut] + dna2[cut:]
        dna2_f = dna2[:cut] + dna1[cut:]

        #mutation process
        dna1_f = error_prbability( dna1_f, prob_err)
        dna2_f = error_prbability( dna2_f, prob_err)


        #number of integers in the list
        N = int(len(dna1_f)/8)
        new_dna1 = []
        new_dna2 = []
        l_b = 0

        #converts the string in a list of integers
        for i in range(N):
                new_dna1.append( int(dna1_f[l_b:8+i*8],2 ) )
```

```
            new_dna2.append( int(dna2_f[l_b:8+i*8],2 ) )
            l_b = 8+i*8


        new_dnas = [ new_dna1, new_dna2 ]


        return new_dnas
```

Appendix C

NetLogo and Python

To link NetLogo and Python it was necessary downloading theextension from the github link: https://github.com/qiemem/PythonExtension. An easy instruction to download it, can be found at page 19 of: guide.

Now we show the few lines of code needed for the connection:

```
;at the start of the code
extensions [ py ]

.
.
.

;in the setup command
py:setup "python3"
py:run "import genetic_algorithm as g_a"

.
.
.

;in the reproduction routine
py:set "DNAs" DNAs
py:set "prob_mutation" prob_mutation      ;mutation probability


set new_DNAs py:runresult "g_a.genetic_algorithm( DNAs, prob_mutation )"
```