UNIVERSITAT POLITÈCNICA DE VALÈNCIA

IN-RED 2019
V Congreso de Innovación Educativa y Docencia en Red

Congreso In-Red 2019
UPV, 11 y 12 de julio de 2019
Doi: http://dx.doi.org/10.4995/INRED2019.2019.10364

A NetLogo® model for introducing students to genetic algorithms*

Juan José Giner-Sanz[1,2], Montserrat García-Gabaldón[2], Emma María Ortega[2], Yang Shao-Horn[1] and Valentín Pérez-Herranz[2]

[1]The Electrochemical Energy Lab, Massachusetts Institute of Technology
[2]Depto. Ingeniería Química y Nuclear, Universitat Politècnica de València

Abstract

The great ubiquity and utility of genetic algorithms (GAs) in nearly every field of Sciences and Engineering, makes them essential for future scientists and engineers. For this reason GAs should be covered in scientific and engineering graduate curricula. In this work, a simple NetLogo® model is presented. Its goal is to introduce GAs in such way that students new to the field can grasp the basic concepts behind GAs while they discover the model. This model may be used in a computer lab session, as an online applet for the students to revise the concepts after the class, or in a Massive Open Online Course (MOOC) course.

Keywords*: Genetic algorithm, Optimization, Control strategy design, Netlogo®, Engineering soft computing techniques.*

Resumen

La gran ubicuidad y utilidad de los algoritmos genéticos (AGs) en casi todos los campos de las Ciencias e Ingenierías, los convierte en una herramienta esencial para los futuros científicos e ingenieros. Por esta razón, los AGs deben incluirse en los planes de estudios de posgrado en Ciencias e Ingenierías. En este trabajo, se presenta un modelo simple de NetLogo®, cuyo objetivo es presentar los GAs de tal manera que los estudiantes nuevos en el campo puedan comprender los conceptos básicos que hay detrás de dichos algoritmos mientras exploran el modelo. Este modelo puede usarse en prácticas informáticas, como una applet en línea para que los estudiantes revisen los conceptos después de clase, o en un curso online masivo y abierto (MOOC, por sus siglas en inglés).

Keywords*: Algoritmo genético, Optimización, Diseño de una estrategia de control, Netlogo®, Técnicas de soft computing en ingeniería.*

*The authors are very grateful to the Generalitat Valenciana and to the European Social Fund, for their economic support in the form of Vali+d postdoctoral grant (APOSTD-2018-001).

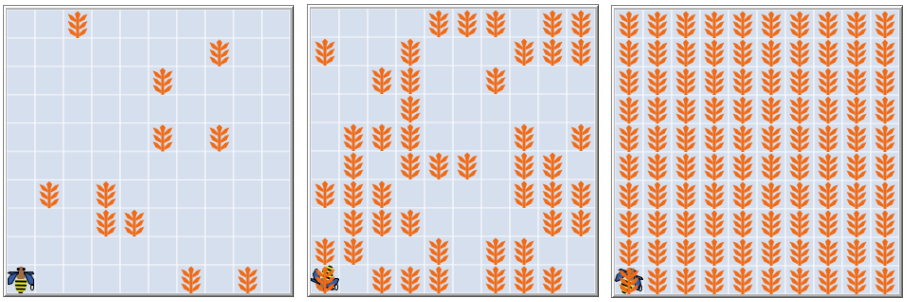
1 Introduction

The idea of an algorithm that emulates biological evolution (i.e. Darwinian evolution) was first proposed by Alan Turing in 1950 (Turing 1950). Shortly after, the first computer simulations of evolution started appearing in literature. First, the 1954 paper (Barricelli 1954) of the Italo-Norwegian mathematician Nils Aall Barricelli. And then, the series of papers (Fraser 1957a; Fraser 1957b; Barker 1958a; Barker 1958b; Fraser 1960a; Fraser 1960b; Fraser 1960c) of the Australian quantitative geneticist Alex Fraser. Fraser's works already included all the essential elements of what today are known as genetic algorithms (GAs). After these germinal works, computer simulation of evolution started gaining importance in the Biology field during the early 1960s.

The first applications of artificial evolution to solving optimization problems are attributed to Barricelli and Bremermann. The first reported the use of an artificial evolution algorithm to optimize the playing strategy for a simple game (Baricelli 1962); whereas the latter, Hans-Joachim Bremermann, a mathematician and biophysicist of the University of California-Berkeley, published a series of papers in which a virtual population was subjected to recombination, mutation, and selection in order to find solutions for optimization problems (Crosby 1973). However, it was not until the works of Rechenberg and Schwefel, in the early 1970s, that artificial evolution became a widely recognized optimization method (Davis 1991).

In this temporal frame, two techniques based on artificial evolution appeared independently: GAs and evolutionary programming (EP). The first, were first proposed in the early 1970s by John Holland as a mean to find good solutions to problems that were otherwise computationally intractable (McCall 2005). The concept originated from the studies on cellular automata that Holland and his students conducted at the University of Michigan. Holland laid the theoretical foundations of GAs with the Holland's Schema Theorem and the related building block hypothesis (Holland 1992). The latter, originally developed by Lawrence J. Fogel, has in common with GAs that they attempt to evolve a string representation through a series of fitness-based evolutionary steps in order to get an optimum solution. Although of independent origin, the two fields have grown together, and today, "evolutionary computation" or "evolutionary algorithms" are sometimes used as an umbrella term for the whole area (Whitley 1994).

In modern computer science, GAs are defined as a family of computational models based on a metaheuristic inspired by Darwinian evolution. The basic idea behind these algorithms is to encode a potential solution of a specific problem using a chromosome-like data structure; and then apply selection, recombination and mutation operators on a population of such chromosomes in order to search for the fittest individual (i.e. the optimum) (Whitley 1994). A typical GA implementation starts with a randomly generated population of chromosomes. Then, the "goodness" of each individual of the population is quantified using the fitness function associated with the specific problem. A selection operator allocates reproductive opportunities in such a way that the chromosomes associated to fitter individuals are given more chances to "reproduce" than the chromosomes associated to lower fitness individuals. After that, recombination (i.e. intensification) and mutation (i.e. exploration) operators are applied in order to obtain the next generation population of chromosomes. This process is repeated a sufficiently large number of times.



(a) $\theta = 0{,}1$ **(b)** $\theta = 0{,}5$ **(c)** $\theta = 1{,}0$

Figura 1: Different environments for RoboBee

Today, GAs have been applied successfully to a very broad range of practical problems in Science and Engineering (Karr y Freeman 1998). Some examples of such problems are the design of a digital communications network (Davis y Coombs 1987), electromagnetic engineering (Johnson y Rahmat-Samii 1997), the AEGIS combat system design (Kuchinski 1985), image enhancement and segmentation (Paulinas y Ušinskas 2007) and molecular modeling (Devillers 1996), amongst many others. The great ubiquity and utility of GAs in nearly every field of Sciences and Engineering, makes them essential for future scientists and engineers. Consequently, GAs should be covered in scientific and engineering graduate curricula.

In this work, a simple NetLogo® model is presented. The goal of this model is to introduce GAs in such way that students new to the field can grasp the basic concepts behind GAs while they discover the model; and they can visualize the role of each one of the GA's parameters as they "play" with the model. In order to illustrate the basic concepts of GAs, the model uses the example of RoboBee, a virtual robotic bee that needs to be programmed for automatically pollinating virtual plants. The model uses a GA to obtain the best strategy for RoboBee. This model may be used in a computer lab session, for the teacher to explain the concepts while the students are "playing" with the model; or as an online applet for the students to revise the concepts after the class, or in a Massive Open Online Course (MOOC) course.

2 RoboBee, the automatic pollinator bee

The example of RoboBee is inspired in Robby, the robot, invented by Melanie Mitchell and described in pages 130 to 142 of her book "Complexity: A Guided Tour" (Mitchell 2009). In turn, Robby was inspired by the Herbert robot developed at the MIT Artificial Intelligence Lab in the 1980s.

RoboBee is a virtual robotic bee that lives in a 2D virtual world in which both, space and time, are discrete. This virtual world consists in a 10×10 grid, enclosed by a wall. This could model a robotic bee placed inside a closed greenhouse. Some of the grid squares contain non-pollinated plants. θ denotes the fraction of grid squares that contain plants. Figure 1 shows examples of different environments for RoboBee.

RoboBee is meant to fly around the world, pollinating the non-pollinated plants. Initially, RoboBee always starts at the bottom left corner of the world (i.e. where its

recharging pod is located). On the one hand, the insect robot has a vision range of 1 square. Therefore, at any given position, it is able to "see" the content of 5 squares: the 4 squares that surround its current square (North, South, West and East) and its current square. There are 3 different options for the content of a given square: empty (or already pollinated plant), non-pollinated plant or wall. On the other hand, in each discrete time step, RoboBee can do one action out of its list of available actions. This lists contains 6 elements: move 1 square North, South, East, West or in a random direction; or execute the pollinating protocol. When the pollinating protocol is executed in a square containing a non-pollinated plant, the plant is pollinated and becomes a pollinated plant (i.e. an empty square for the robot). On the contrary, when the pollinating protocol is executed in a square that does not contain a non-pollinated plant, the pollination tool may get damaged. Another potential hazard for the robot is bumping into walls. Because of this, both, bumping into walls and executing the pollinating protocol in a square which does not contain a non-pollinated plant, should be avoided.

The goal is to develop a strategy (i.e. control program) for RoboBee, so that the robot pollinates the maximum number of plants while avoiding to bump against walls and misexecuting the pollinating protocol. In order to quantify the efficiency with which the robot performs its job, the following system of rewards and penalties was considered:

- For each non-pollinated plant that is successfully pollinated: +10 points.
- For each bump against a wall: -5 points.
- For each execution of the pollinating protocol in a square which does not contain a non-pollinated plant: -1 point.

Given a certain environment and a given strategy, the score of the robot is defined as the sum of the rewards and penalties obtained by the robot after N_m movements in the considered environment according to the given strategy.

In order to fulfill the goal (i.e. develop the control strategy of the RoboBee) two approaches will be considered. On the one hand, students will design "by hand" a control strategy for the robot. An example of the control strategies that students could propose, is:

> *If there is a non-pollinated plant in my current square*
>
> *Then Execute the pollinating protocol*
>
> *If there is a non-pollinated plant in a neighbor square*
>
> *Then Move to that square*
>
> *Otherwise Move randomly*

On the other hand, a GA will be used in order to design the control strategy without any human intervention in the design process. Then, the efficiency (i.e. score) of the

Cuadro 1: Structure of RoboBee's rule matrix

	Inputs					
Case Id.	N	S	E	W	C	Output
1	NP	NP	NP	NP	NP	Move N
2	NP	NP	NP	NP	P	Move E
⋮	⋮	⋮	⋮	⋮	⋮	⋮
243	W	W	W	W	W	Move E

strategy proposed by the students will be compared to the one of the strategy designed by the GA.

A strategy can be defined using words, as in the aforementioned example; or it can be defined as a rule matrix. A rule matrix consists in a matrix that lists the output action for each one of all the possible input combinations. Since there are 5 inputs (N, S, E, W and C) and 3 possibilities for each input (No plant, NP; Plant, P; Wall, W), then the rule matrix will have 243 (i.e. 3^5) rows: it will give the output (i.e. action performed in that situation) for each one the cases. Table 1 shows the structure of a rule matrix for RoboBee. For instance, the first row of that matrix means that if there are no non-pollinated plants in any of the 5 squares the bee can see, then the bee moves north. It should be noted that not all the cases in the matrix are possible: for example, case 243 is not possible, since there cannot be a wall simultaneously in the 5 squares the bee can see. However, for programming convenience, these cases are not removed from the rule matrix.

Since there are 6 possible actions, the space of possible strategies contains 6^{243} candidates. It is obvious that a brute force method (i.e. try all the possible strategies, one after the other) is not possible in this case. In order to be able to run a GA the chromosome structure must be defined. The first step is to numerically code the 6 possible actions: a number between 0 and 5 can be assigned to each action. The chromosome (i.e. strategy) consists in the list of the numerical codes of the actions that the robot will perform in each one of the 243 cases defined in the rule matrix (in the order defined in the matrix). In other words, the chromosome will be a vector of 243 elements, each element being a integer number between 0 and 5: the chromosome consists in 243 genes with 6 possible alleles each one. Since in this case all the genes have the same alleles, it is not necessary to convert the chromosome to a binary chromosome; and the decimal one can be used as is in the GA.

After defining the chromosome, the next step before being able to run a GA, is defining the fitness function. In this case, the fitness function was defined as the average score of the strategy after testing it in N_e randomly generated environments.

The GA can then be applied:

1. An initial population of N_p random chromosomes (i.e. vectors of 243 elements between 0 and 5) is generated.

2. The fitness of each chromosome is evaluated by simulating the corresponding strategy in N_e randomly generated environments, and averaging the scores obtained in each simulation.

3. The offspring generation is generated by applying a reproduction operator followed by a mutation operator to the parental population. The reproduction operator (i.e. selection and recombination) is defined by the tournament size, N_t: only the N_t best individuals of the population are selected for breeding. The reproduction is done by randomly selecting two parents of the breeding pool (with a probability proportional to their fitness), and performing a crossover of their chromosomes around a randomly selected crossover point. The mutation operator replaces with a mutation probability of λ_m, a random element in the children' chromosome by a random number between 0 and 5.

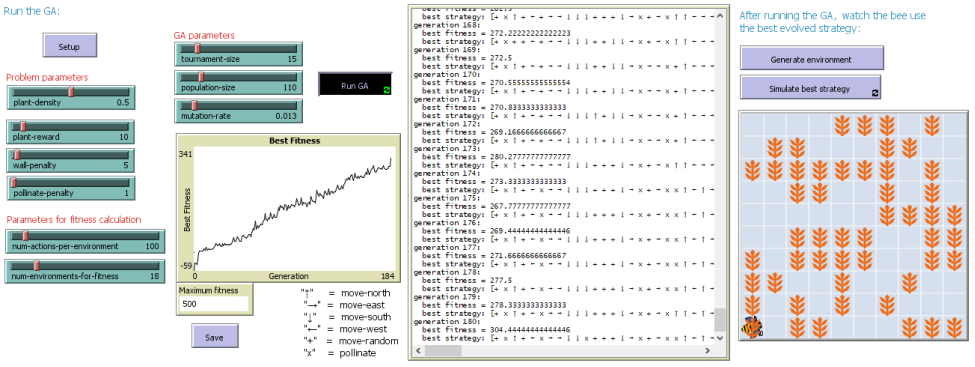
4. Once an new population of N_p children has been obtained, the process is repeated from step 2.

3 The NetLogo® model

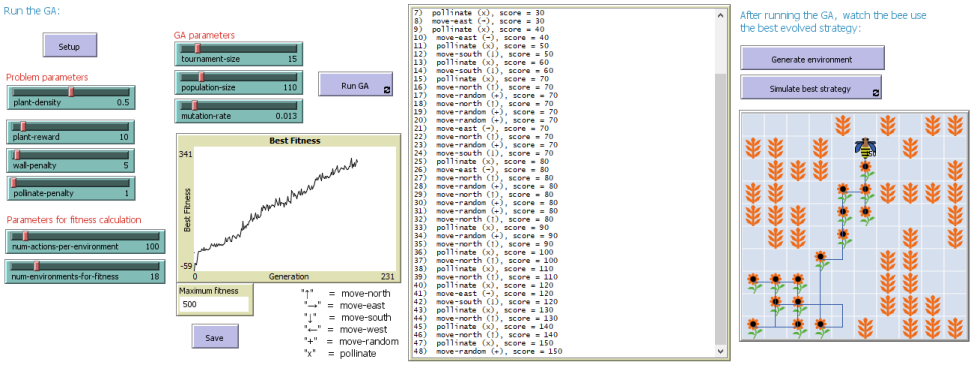
NetLogo® is a multi-agent programmable modeling environment, designed by Uri Wilensky, that can be downloaded for free from its official web page (Wilensky 1999). The NetLogo® model presented in this work is based on Melanie Mitchell's NetLogo® model of Robby, the robot (Mitchell, Tisue y Wilensky 2012). Figure 2 shows the front panel of the model. The user interface allows the user to select the problem parameters (the plant fraction, θ; and the rewards and penalties), the fitness calculation parameters (the numbers of environments each strategy is tested in, N_e; and the number of movements simulated in each environment, N_m), and the parameters of the GA (the population size, N_p; the tournament size, N_t; and the mutation rate, λ_m).

During the execution of the GA (figure 2a), the best fitness plot is updated in real time: after each generation is calculated, the fitness of the best individual of the generation is represented on the plot. This representation is useful for identifying when the GA has converged. Furthermore, in the dialog box (i.e. the middle text box), both, the best chromosome and its score, are displayed after each generation is calculated. Once the GA has been run, the strategy associated to the best chromosome of the final population can be simulated (figure 2b). For that simulation, a new random environment can be generated, and then, the actions of the RoboBee can be observed: the path of the robot (and how it pollinates plants) can be followed in the bottom right diagram, while the different actions of the bee are displayed in the central dialogue box. The numeric label next to the bee represents the score of the bee at that particular time step.

The model requires a relatively high amount of time to converge when run on a 2018 state-of-the-art laptop. For this reason, when using it in the context of a computer lab practice, it would be more interesting to present the model during the class and then have the students run the model in order to study the effect of the different parameters as a homework assignment, rather than expecting to fully run the model during the class.



(a) Running the GA



(b) Simulating the best strategy

Figura 2: Front panel of the RoboBee NetLogo® model

4 Results and discussion

One of the things that students can study using the NetLogo® model presented in this work, is the effect of the GA parameters. Figures 3 and 4 show the fitness plots obtained for different values of the GA parameters, for $\theta = 0{,}5$. These plots are the representation of the population fitness for each generation. Figure 3 studies the effect of the population size, N_p; whereas figure 4 studies the effect of the mutation rate, λ_m. In these figures, the red dashed line indicates the maximum achievable fitness (i.e. based on the total number of plants in the environment and the reward for pollinating one of them); while the gray dashed line marks the average fitness obtained by the human-designed strategy example described in section 2.

All the presented fitness curves display the same overall shape: a relatively noisy curve, that initially increases with the generation number, and finally converges to a final fitness. On the one hand, the initial increase of the fitness curve is due to the fact that, initially, the best chromosome of the population gets better and better (i.e. higher fitness) thanks to the guided evolution process. On the other hand, the noise is generated by the fact that the fitness function is a stochastic function, since the environments in which the strategy is tested are generated randomly. An easy way to reduce the noise variability is to increase the number of environments in which each strategy is tested in.

In figure 3 it can be observed that for the 3 considered values of N_p, the fitness curves converge to the same final fitness, but they do it at different convergence rates. Therefore, the population size does not affect the chromosome the GA converges to, but it does affect the speed (i.e. required number of generations) at which it converges. For higher N_p the GA converges faster: it requires less generations to reach its final fitness. This is due to the fact that for larger N_p, more offspring chromosomes are generated in each generation (i.e. more cases are explored in each generation). The effect of N_p on the convergence rate is not linear: For instance, increasing N_p from 55 to 110 (factor 2 increase) reduces the number of generations required to reach the final fitness from nearly 2000 to a little bit below 500 (factor 4 reduction); while increasing N_p from 110 to 220 (factor 2 increase) reduces the number of generations required to reach the final fitness from around 500 to nearly 250 (factor 2 reduction). Increasing the population size increases proportionally the computational time required for each generation. Consequently, there is an optimum population size, which in this case is around 110.

In figure 4 it can be observed that the mutation rate has a strong effect on the final fitness value, though the problem parameters remain the same (i.e. the optimum strategy is the same in the 3 cases). On the one side, in the case $\lambda_m = 0$ (i.e. no mutation operator), the fitness improves slightly in the first generations and then does not improve further. This is because removing the mutation operator eliminates the exploration function of the GA, that then only performs its intensification function. Without the diversification generated by the mutation operator, an endogamy problem arises and causes a premature convergence (i.e. does not converge to the optimum solution) of the GA. On the other side, in the case $\lambda_m = 0{,}05$ (i.e. high mutation rate), the high mutation rate causes disruptive effects: the traits that lead to an evolutive advantage are lost within a few generations due to the great number of mutations. Consequently, there is an intermediate optimum mutation rate that should be big enough to avoid the endogamy problem, but at the same time, should be small

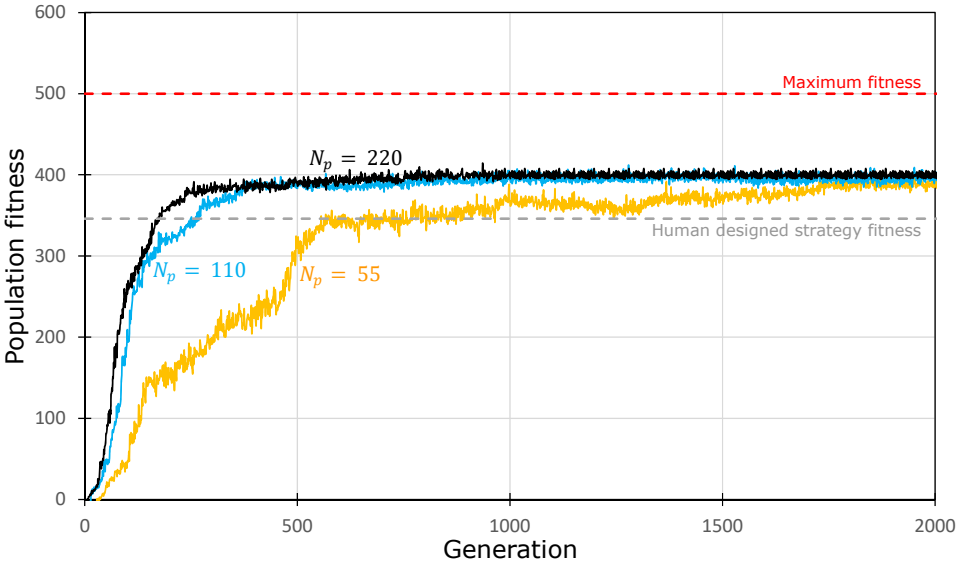


Figura 3: Effect of the population size

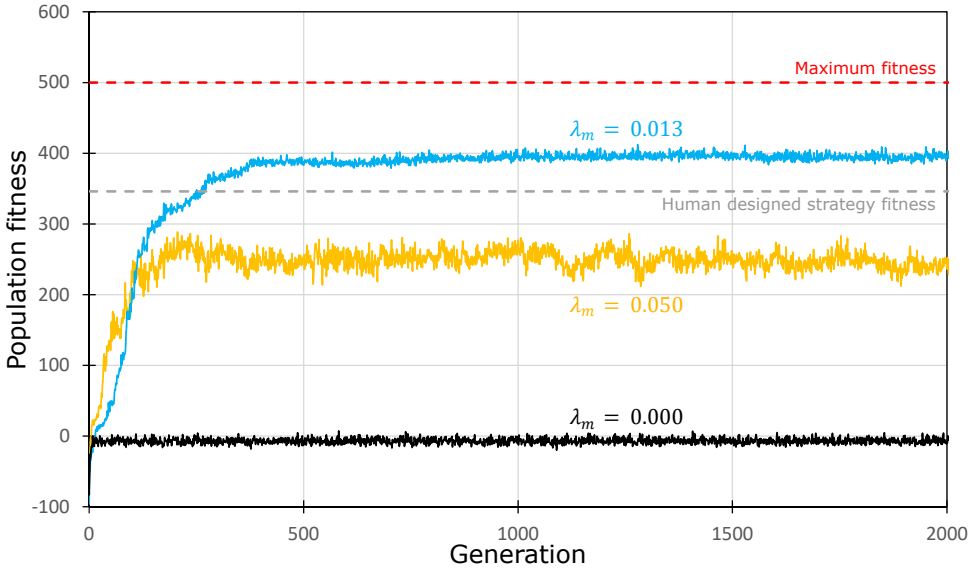


Figura 4: Effect of the mutation rate

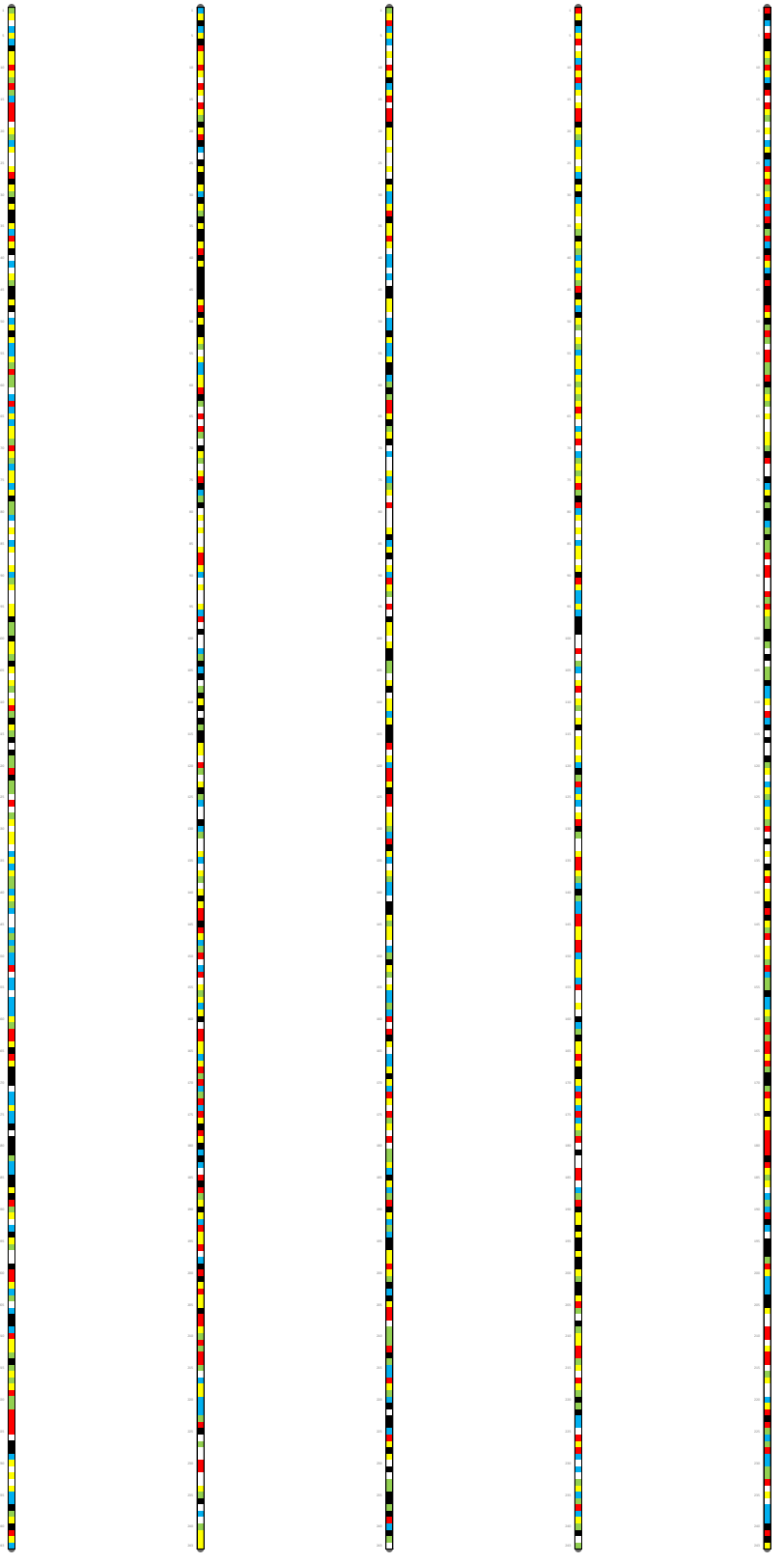
enough to avoid disruptive effects due to mutations. In this example, the optimum mutation rate is around 0.013.

Another thing that students can study using the NetLogo® model presented in this work, is the effect of the problem parameters (i.e. plant fraction, rewards and penalties) on the optimum strategy for RoboBee. For instance, figure 5 gives the optimum chromosome for different plant fractions. In the aforementioned figure, the different alleles are encoded using color: move North (white), move East (black), move South (red), move West (blue), move randomnly (Green) and execute the pollinating protocol (yellow).

Comparing the chromosomes with each other, 3 types of chromosomatic regions can be identified: invariable regions, that remain constant with θ; high variability regions, that are different for every θ; and θ-trend regions, that present a gradual change with θ. An example of the first type is gen 10, which optimum allele is move South for every θ. Gen 10 corresponds to the situation in which there is a non-pollinated plant in the South square, and all the other visible squares are empty (or contain an already pollinated plant). The results indicate that in any environment (i.e. for any θ) the optimum action in the aforementioned situation is move South. An example of the second type is gen 243, which optimum allele changes randomly from one θ to another. In general, type 2 gens are useless gens (i.e. gens that encode non-possible situations, as discussed in section 2), and therefore the optimum allele found for such gens is purely random since it has no real effect on the fitness of the strategy. For instance, gen 243 corresponds to the situation in which there is a wall in the 5 visible squares, which is obviously not possible! Finally, gen 2 is an example of the third type: its optimum allele in execute the pollinating protocol for θ between 0.2 and 0.8, and changes to move East for $\theta = 1$. This gen corresponds to the situation in which the current square contains a non pollinated plant, while the other 4 visible squares are empty. This results indicate that in environments with low and moderate plant fractions, the best option in the aforementioned situation is to execute the pollinating protocol; but in fully planted environments, that is no longer the case. In short, type I regions form the basic scaffold of the optimum strategy, which does not depend on the plant fraction. This scaffold is nuanced by type III regions, that carry the plant fraction dependent parts of the strategy. Finally, type II regions are just useless regions of the genome.

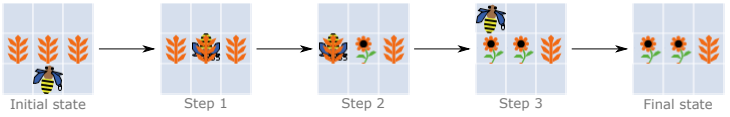
Similarly to the above analysis, students could use the NetLogo® model presented in this work, in order to analyze the effect of the rewards and the penalties. For instance, they could study how the optimum strategy would change if much more importance was given to wall crash avoidance (i.e. higher wall penalty).

A final thing students can reflect on using this model is why the GA-designed strategy performs better than the human-designed strategy, as it can be observed in figure 3. These reflections can make students improve their design skills. For instance, one of the features that makes the GA-designed strategy win the human-designed strategy, is the emergence of a memory-like mechanism, illustrated in figure 6. In this simple way, the GA-designed strategy is able to implement a memory, in a robot that actually has no memory.



(a) $\theta = 0{,}2$ **(b)** $\theta = 0{,}4$ **(c)** $\theta = 0{,}6$ **(d)** $\theta = 0{,}8$ **(e)** $\theta = 1{,}0$

Figura 5: Best chromosome for different environments



(a) Human-designed strategy

(b) GA-designed strategy

Figura 6: Emergence of a memory-like mechanism in the strategy designed by the GA

5 Conclusions

In conclusion, the NetLogo® model presented in this work can be used to present and illustrate, using a real application example, the basic concepts of GAs. By using the model, students can achieve multiple outcomes, some of which are:

1. Understand how a real application example can be modeled in order to apply a GA on it.
2. Study the effect of the GA parameters.
3. Analyze how the optimum strategy changes when the problem settings change.
4. Identify interesting strategic features that emerge from the GA.

This program may be used in a computer lab session; or as an online applet for the students to revise the concepts after the class, or in a Massive Open Online Course (MOOC) course. Due to its computation time requisites, when using the model in the context of a computer lab practice, it is more interesting to present the model during the class and then have the students run the model as a homework assignment, rather than expecting to fully run the model during the class.

References

Baricelli, N.A. (1962). "Numerical testing of evolution theories, part II preliminary tests of performance". En: *Symbiogenesis and Terrestrial Life, Acta Biotheoretica* 16, págs. 99-126.

Barker, J.S.F. (1958a). "Simulation of genetic systems by automatic digital computers". En: *Australian Journal of Biological Sciences* 11.4, págs. 603-612.

— (1958b). "Simulation of genetic systems by automatic digital computers. IV. Selection between alleles at a sex-linked locus". En: *Australian Journal of Biological Sciences* 11.4, págs. 613-626.

Barricelli, N.A. (1954). "Esempi numerici di processi di evoluzione". En: *Methodos* 6.21, págs. 45-68.

Crosby, J.L. (1973). *Computer simulation in genetics*. London: John Wiley & Sons.

Davis, L. (1991). *Handbook of genetic algorithms*. New York: VNR Computer Library.

Davis, L. y S. Coombs (1987). "Genetic algorithms and communication link speed design: theoretical considerations". En: *Genetic algorithms and their applications: Proceedings of the second International Conference on Genetic Algorithms: July 28-31, 1987 at the Massachusetts Institute of Technology, Cambridge, MA*.

Devillers, J. (1996). *Genetic algorithms in molecular modeling*. Academic Press.

Fraser, A.S. (1957a). "Simulation of genetic systems by automatic digital computers. I. Introduction". En: *Australian Journal of Biological Sciences* 10.4, págs. 484-491.

— (1957b). "Simulation of genetic systems by automatic digital computers II. Effects of linkage on rates of advance under selection". En: *Australian Journal of Biological Sciences* 10.4, págs. 492-500.

— (1960a). "Biometrical Genetics". En: ed. por O. Kempthome. New York: Macmillan. Cap. Simulation of genetic systems by automatic digital computers. V. linkage, dominance and epistasis, págs. 70-83.

— (1960b). "Simulation of genetic systems by automatic digital computers. VI. Epistasis". En: *Australian Journal of Biological Sciences* 13.2, págs. 150-162.

— (1960c). "Simulation of genetic systems by automatic digital computers VII. Effects of reproductive rate, and intensity of selection, on genetic structure". En: *Australian Journal of Biological Sciences* 13.3, págs. 344-350.

Holland, J.H. (1992). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. Cambridge: MIT press.

Johnson, J.M. y V. Rahmat-Samii (1997). "Genetic algorithms in engineering electromagnetics". En: *IEEE Antennas and Propagation Magazine* 39.4, págs. 7-21.

Karr, C. y L.M. Freeman (1998). *Industrial applications of genetic algorithms*. New York: CRC Press.

Kuchinski, M.J. (1985). *Battle management systems control rule optimization using artificial intelligence*. Technical Note. Dahlgren, VA: Naval Surface Weapons Center.

McCall, J. (2005). "Genetic algorithms for modelling and optimisation". En: *Journal of Computational and Applied Mathematics* 184.1, págs. 205-222.

Mitchell, M. (2009). *Complexity: A guided tour*. Oxford: Oxford University Press.

Mitchell, M., S. Tisue y U. Wilensky (2012). *NetLogo Robby the Robot model*. http://ccl.northwestern.edu/netlogo/models/RobbytheRobot. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.

Paulinas, M. y A. Ušinskas (2007). "A survey of genetic algorithms applications for image enhancement and segmentation". En: *Information Technology and Control* 36.3, págs. 278-284.

Turing, A.M. (1950). "Computing machinery and intelligence". En: *Mind* LIX.236, págs. 433-460.

Whitley, D. (1994). "A genetic algorithm tutorial". En: *Statistics and Computing* 4.2, págs. 65-85.

Wilensky, U. (1999). *NetLogo*. http://ccl.northwestern.edu/netlogo/. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.