Intelligent Systems

Genetic Algorithms

Miguel Jiménez Benajes
Computer Engineering, UJI
Castellón, España
migueljimenezbenajes@gmail.com

***Abstract*— Genetic algorithms are framed within the evolutionary algorithms and constitute one of the main branches within the intelligent systems. Following the principles of Darwin's evolutionary theory, these algorithms have been able to respond to problems that other algorithms have not been able to solve. This document is a brief introduction and an exposition of personal and laboratory work on the subject.**

I. Introduction

Figure 1. Evolution parody with genetic algorithms at the top [1].

The concept of machine learning is strongly rooted in the concept of genetic algorithm. It is for this reason that Alan Turing is mentioned, in 1950 he published an academic article on artificial intelligence, "Computing Machinery and Intelligence" where the famous "Turing Test" is exposed and it is asked if the machines are able to think.

In parallel, in 1954, Nils Aall Barricelli began computer simulation of evolution. Although it did not obtain great recognition, it laid the bases of the genetic algorithms.

Slowly and throughout the 60's the simulation of evolution by computer was becoming more common among biologists. The methods used by Fraser and Burnell were published in 1970 and contained all the basic elements of the current evolutionary algorithms.

In 1975, John Henry Holland (a pioneer in complex systems and non-linear science) published one of the most important books in the field. "Adaptation in Natural and Artificial Systems". This publication is the beginning in the field of automated learning for the design of complex devices such as aircraft turbines, integrated circuits, satellite antennas etc.

Genetic algorithms are currently applied to optimization problems in very complex or non-derivable functions. Its use goes from design of systems of climatology to economy, through design of video games.

II. Darwin's Theory Of Evolution

The roots of evolutionary algorithms rest on Darwin's theory of evolution. This corpus, published in "The Origin of Species" (1859), maintains that living beings had a common origin and, due to various factors, have evolved over millions of years to become the species that we know now .

This theory starts with natural selection. The survival and reproduction of an individual of a species is conditioned to the environment and its genotype. So only individuals with more adapted genetic material will survive.

"It is not the strongest of the species that survives, nor the most intelligent, but the one most responsive to change."[2].

III. Biological vs Computational Elements

As genetic algorithms are inspired by biology, there are similar elements among them [3].

Element	**Biological**	**Computational**
Allele	"Value" of a gene.	Value of a information object.
Chromosome	DNA molecule with genetic material.	Sequence of information objects.
Fitness	Aptitude of a living organism that determines the chances of survival and reproduction.	Quality of a solution candidate that determines the chances of survival and reproduction.
Gene	Part of a Chromosome, as a fundamental unit of inheritance.	Information object, as a fundamental unit of inheritance.
Generation	Population at a point in time.	Population at a point in time.
Genotype	Genetic constitution of a living organism.	Encoding of a solution candidate.
Individual	Living organism.	Solution candidate.
Phenotype	Physical appearance of a living organism.	Implementation of a solution candidate.
Population	Set of living organisms.	Set of Chromosomes.
Reproduction	Crating offspring of one or multiple organisms.	Creating chromosomes from one or multiple chromosomes.

IV. Genetic Algorithm Process

In figure 2 we can observe the general and simpler scheme by which the genetic algorithms are governed.

- Initialise Population. At first the population starts with a limited number of individuals. Each of them with a different genotype and randomly generated normally.
- Evaluation. Subsequently the genotypes are punctuated with an evaluation function previously created to reach the expected result. Generally the better the evaluation function, the better results we will get.
- Selection. Once rated, those who have obtained better scores are selected.
- Crossover. The best chromosomes are crossed to obtain new individuals (similar to reproduction). In this process two individuals (parents) are usually used to obtain two children. There are several variants of this process, but the most common are single-point crossover (Fig. 3) and two-point crossover (two cut-off points are randomly selected) (Fig. 4).
- Mutation. The mutation process adds random changes to the chromosomes of the new individuals, in this way, the GA can get a better solution.
- End of the algorithm. Finally, the process is repeated from the evaluation if the criterion of completion has not been met. In the end we get a set of evolved individuals.

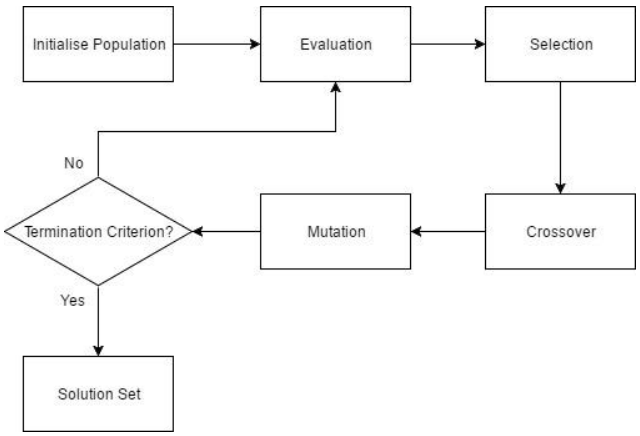


Figure 2. General genetic algorithm process. [4]

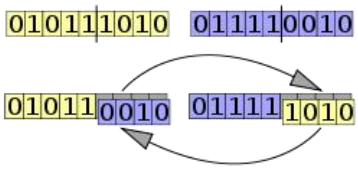


Figure 3. Single-point crossover. [5]

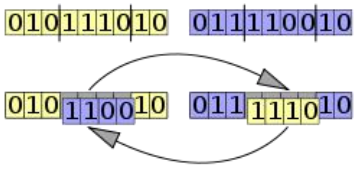


Figure 4. Two-point crossover. [5]

V. Fourth lab Session: Genetic Algorithms

In the fourth session of the laboratory, the concepts learned in the theory session on genetic algorithms were reviewed. The Pyevolve environment, an open-source framework written in python, allows us to approach the world of genetic algorithms in a simple, comfortable and transparent way.

During the session four simple problems were performed in order to introduce us to the development environment.

- The first problem was to find a list of zeros in all positions of size "n". Although the problem is very simple, it is very useful to test the environment. After a few tests, for a list of 20 integers, pyevolve finds the solution in about 16-19 generations.
- Graphical analysis. Pyevolve brings a graphical analysis tool based on the "Matplotlib" library. Using the above-mentioned problem, a deeper graphic analysis of the solution is performed. In Figure 5 we can see how in each generation the average scores obtained by the population increase.
- Rastrigin function (fig. 6). The rastrigin function is generally used to test optimization algorithms. It is very suitable because it contains a large number of local minimums, so it serves to prove the ability of the algorithm to avoid these minimums and obtain the global minimum. Genetic algorithms are used very frequently in optimization problems, which is why it has been tried to evaluate rastrigin function with pyevolve. As can be seen in the analysis of figure 7, the algorithm complies with the typical behavior (fig. 8) of genetic algorithms in minimization problems.
- Traveling Salesperson Problem. The problem of the traveling salesperson is a typical NP-Hard problem, which means that it is a problem that is at least as complex as an NP, so that (for the moment) it can not be solved in a polynomial time. For this type of problems genetic algorithms are very useful, since they allow us to find a good solution in a reasonable time. In the laboratory session the code was modified to use a heavy graph containing 48 cities. After modifying some parameters, the best score obtained was 36723, a little far from the optimal cost (10628).

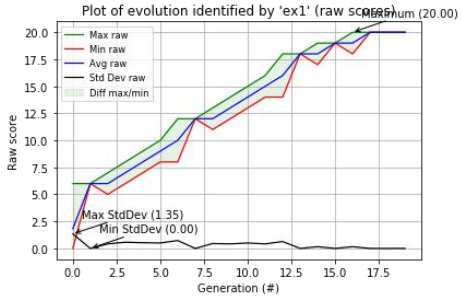


Figure 5. Max/min/avg/std. dev. Graph (raw scores).

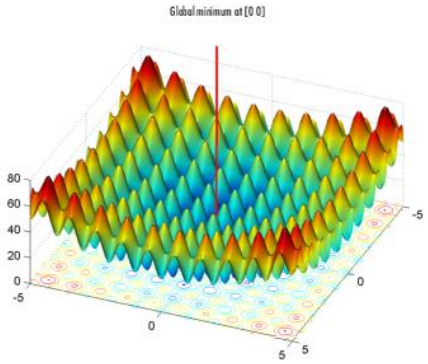


Figure 6. 3D plot of Rastrigin's function. [6]

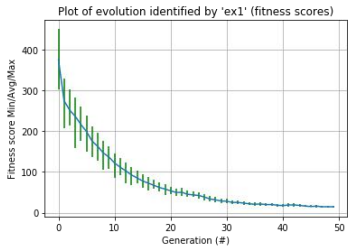


Figure 7. Error bars graph (fitness scores).

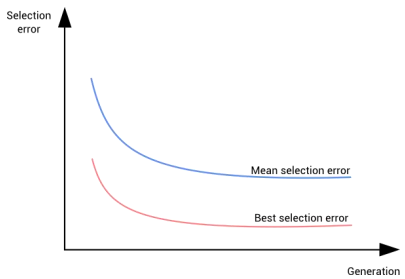


Figure 8. Typical behavoir of the genetic algorithm. [7]

VI. Fifth Lab Session: Robot Crawling

The fifth lab session consisted of applying the knowledge acquired in the previous laboratory session to a more practical and tangible environment. It made use of a virtual environment based on UDK (a 3D engine designed for video games) adapted to the robot NAO. This robot created by SoftBank robotics is a programmable humanoid intended for teaching and research.

In the session the genetic algorithm of pyevolve was used so that the robot learned to crawl. An initial population of about 10 individuals was created that contained a periodic function for simple oscillatory movements. The 12 oscillators (the function) contained, were evaluated at each joint of the robot (fig. 9) to calculate its movement at a certain instant in time. These 12 oscillators make a total of 24 parameters, removing some that are redundant, make a total of 19 parameters. These parameters constitute the genome of each individual and are randomly generated at the beginning.

The most complicated process was to create an evaluation function that allowed to scale the parameters of the robot to those of a predefined range so that the articulations would not be turned of thread. The function returned the distance traveled by the robot. Subsequently, a conditional sentence was added so that, in the event that the robot turned upwards, it returned to its crawling position.

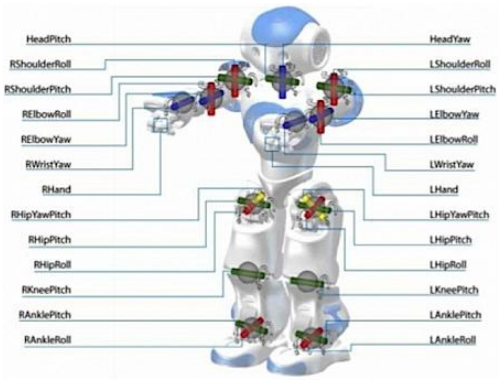


Figure 9. NAO robot articulations. [8]

VII. Genetic Algorithm in Unity 3D

Continuing with the UNITY 3D environment, a basic general-purpose genetic algorithm [9] has been developed in C # in order to go deeper into the subject and to be used in video game AIs. The algorithm contains all the basic concepts seen in class and proper to the genetic algorithms, its scheme is the same as that of figure 2.

The main parameters it receives are the number of individuals in the population, chromosome length, mutation rate and crossover rate. In addition, in each generation the average of scores obtained with the evaluation function can be visualized. The most basic methods used in each new generation are described below.

- calculateBestWorstAvTot(). This method updates data on population scores. Total scores, average scores, best score and worst score.
- getChromoRoulette(Genome[] pop). This method used in the reproduction stage makes use of the roulette selection system also used in laboratory session 5. In this way, individuals with higher scores are more likely to be chosen.

- epoch(List<Genome> oldPopulation). This is the main method used for each generation change. It receives a list of genomes that represent each individual in the population and results in a list of genomes applying the selection, crossover, and mutation processes.

VIII. Neuroevolution: Bacterium

The concept of neuroevolution was born in the 90s and combines machine learning with evolutionary algorithms. Its use applies to video games, artificial life and evolutionary robots.

To implement this concept, an environment has been created in UNITY to evolve a population of virtual "bacteria" [10] [11]. The basic idea is that bacteria learn to obtain the food that is in the two-dimensional space (Figure 10). Each bacterium contains a neural network composed of only 2 perceptrons with a total of 4 inputs and 2 outputs (Figure 11). The genome of each individual is composed of the number of weights of the neural network, for this case a total of 10 pesos, including the bias. The inputs correspond to the X and Y coordinates of the direction of the bacteria and the nearest food. The outputs correspond to the left and right movement. Each time a bacterium gets food, it teletransports to another position in the plane and the bacteria increase its punctuation, so that the ones that get more food will be more likely to reproduce.

Graph 1 shows the average of scores in relation to the number of generations for a population of 30 individuals, a mutation rate of 0.4, a crossover rate of 0.7 and 60 of food. As you can see, the average score increases with each generation. As generations pass, as in most evolutionary algorithms, progress is less.

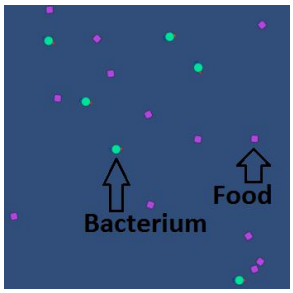


Figure 10. Detail of two-dimensional environment.

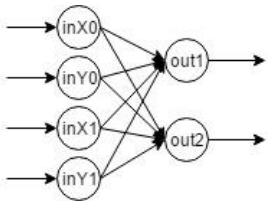
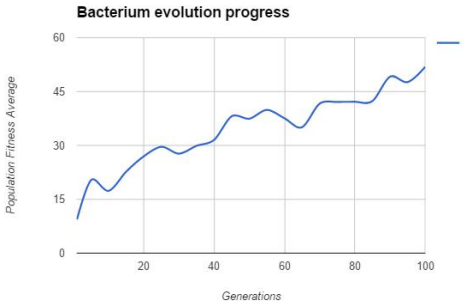


Figure 11. Neural Network scheme used in bacterium.



Graphic 1. Bacterium evolution progress.

IX. Neuroevolution: Flappy Bird

Flappy Bird is a popular game developed by an independent programmer who became the most downloaded within the mobile app stores. The game consists of going by drawing a few tubes jumping with a bird pressing the screen to reach the best score.

In Figure 12 we can see a copy of the Flappy Bird style in UNITY [12]. The basic idea is to evolve a neural network so that the airplane learns to dodge the rocks [13]. For this case, a neural network composed of 3 inputs, 1 hidden layer of 4 neurons and 1 output has been used. Therefore, the genome of each individual will be composed of 21 pesos. The inputs correspond to the Y coordinate in the two-dimensional plane of the plane and to the X and Y coordinates of the vector toward the nearest rock. The output in this case is a number between 0 and 1, if it is greater than 0.5 indicates that it should jump and if it is less than or equal does not jump. The criterion of generation change in this case is fulfilled when all individuals are dead. To evaluate each individual simply calculate the distance he has traveled.

In graph 2 the mean scores (blue line) are shown together with the maximum scores (red line) in relation to the number of generations for a population of 30 individuals, 0.3 mutation rate and 0.7 crossover ratio. From generation 29 there were already two individuals who dodged all obstacles without fail and therefore there was no change of generation. The graph does not follow the typical progress that evolutionary algorithms, however, is proven to work properly.

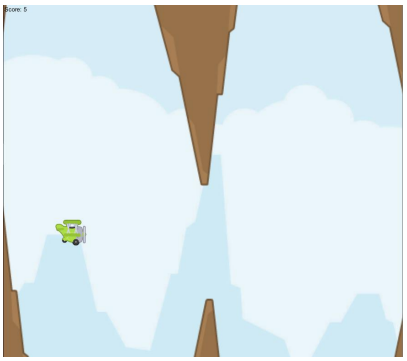
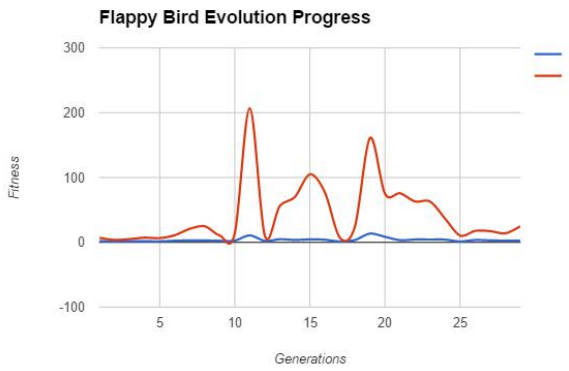


Figure 12. Flappy Bird in UNITY 3D.



Graphic 2. Flappy Bird evolution progress.

X. CONCLUSIONS

Perhaps this is one of the most interesting branches from my point of view in the field of artificial intelligence. The discovery of genetic algorithms opened new doors to knowledge and even today it is fascinating how they manage to solve problems that other algorithms can not.

Lab classes have served me well and I found them very interesting. While it is true that I think we should touch the subject a little more closely and tackle more code instead of working on environments with Pyevolve already created. The fifth session has inspired me to investigate the subject of neuroevolution, an issue I was completely unaware of.

REFERENCES

[1] http://www.genetic-programming.com

[2] Charles Darwin.

[3] https://www.semanticscholar.org/paper/Darwin-or-Lamarck-Future-Challenges-in-Holzinger-Palade/c50cabdbad4af1d411dc7e63b8eb17c749989dce

[4] http://www.ecs.umass.edu/mie/labs/mda/mechanism/papers/genetic.html

[5] https://es.wikipedia.org/wiki/Recombinaci%C3%B3n_(computaci%C3%B3n_evolutiva)

[6] https://es.mathworks.com/help/gads/example-rastrigins-function.html

[7] https://www.neuraldesigner.com/blog/genetic_algorithms_for_feature_selection

[8] http://ima.nyu.sh/introduction-to-robotics/

[9] https://gist.github.com/mijim/1d75823d6ab6cec4c8d7f783215d1ccd

[10] http://www.ai-junkie.com/ann/evolved/nnt1.html

[11] https://gist.github.com/mijim/b3dc84b18d1dcaec5b79c05b79529995

[12] http://anwell.me/articles/unity3d-flappy-bird/

[13] https://gist.github.com/mijim/1b7dcd2549aa4ebce6745b1d0404ad21