Genetic Algorithms

Borja Jiménez, Marcos García

Jaume I University, Spain

al361929@uji.es

al375909@uji.es

***Abstract*— Genetic algorithms take a relevant place inside the field of machine learning due the over performance generating high-quality solutions to optimization problems. For this reason, we have implemented a genetic algorithm based in the Pyevolve library as an evolutionary strategy to develop an automatic generation of movements on the NAO Robot that could mimic a human crawling. The results were run on a simulation environment powered by the Unreal Engine. After 50 generations the training session was completed, lending on a suitable configuration of the robot's parameter and allowing it to execute a proper set of movements capable of crawling on a straight line.**

***Keywords*— Genetic Algorithms, Natural evolution, Mutation, Population, Optimization**.

I. Introduction

To understand what are the genetic algorithms we have to go many years back in time when Charles Darwin shared his theory of the biological evolution. Charles defined the evolution of the species as "offspring with modification". The main idea behind that is species mutate randomly over time. Which it may or may not be beneficial. If the mutation improves some specific quality of an individual that could yield in a better adaptation in the environment, been more likely to survive and reproduce bequeathing the mutation to his offsprings. The sum of favourable mutations rises to the origin of new species that share a common ancestor.

Genetic algorithms represent this process of natural selection where the fittest individuals from a population are selected to produce an offspring for the next generation [1]-[2]. These algorithms are used to generate high-quality solutions to complex problems related to biology like mutation, crossover or selection [3].

The first person who develops this idea was John Henry Holland in 1975 in his book "Adaptation in natural artificial systems". *"He described how to apply the principles of natural evolution to optimization problems and built the first Genetic Algorithms."* [4]

The remaining sections of this paper are as follows: Sections 2 and 3 introduce basic concepts to understand the functioning of Genetic Algorithms.

Section 4 shows how the experiments have been performed and what methods and materials were used.

Section 5 describes how the experiments were implemented and studied to understand how genetic algorithms work.

Section 6 ends the article with some conclusions about the study.

II. Darwin evolution

In nature, the individuals of a specie compete with other species for resources like food, water, etc. Even some individuals of the same specie compete in the search for partner or territory. Those individuals who achieve this, are more likely to generate a large number of offsprings. However, less gifted individuals will produce few descendants. That means that the genes of the best individuals will spread to the next generations. The combination of good modifications from different ancestors can sometimes produce a well-fitted individual that have a better adaptation than his ancestors [5]-[6].

This process is known as “Natural selection” and we are going to explain it better with an example. As we can see in Figure 1, a population of rats have moved to a new area where the rocks are darker. Due to the natural genetic variation, some individuals are darker than the others. These darker individuals are less visible for the birds so they have more chances to survive. For this reason, the next generation will have more individuals with dark hair than the previous generation.

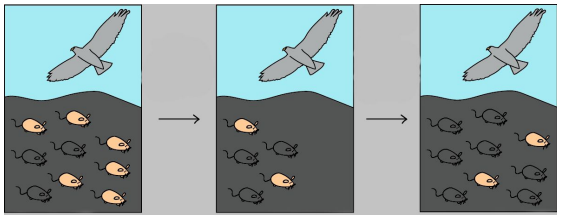

Fig. 1 Natural selection where the grey colour of the hair causes an advantage over the brown colour..

III. Genetic Algorithms

Genetic Algorithms try to mimic the behaviour of nature. They work creating a population of the individual which represents the abstraction of the problem that it is meant to be solved and optimized. Each individual has assigned a score, that represents the goodness of its performance. In nature, it would work as a representation of how well this specific individual fits in the environment. The greater the adaptation of an individual to the problem, the greater the probability that it will be selected to reproduce, crossing its genes to other individual selected with the same rules, and passing the gene material on to future generations. The variation produced by crossing different individuals yields into new individuals who share some of the characteristics of their parents and extra ones due to mutations. Following this logic, the characteristics that produce an individual overperforms will spread through the population. If the Genetic Algorithm has been well designed, the population will converge towards an optimal solution to the problem.

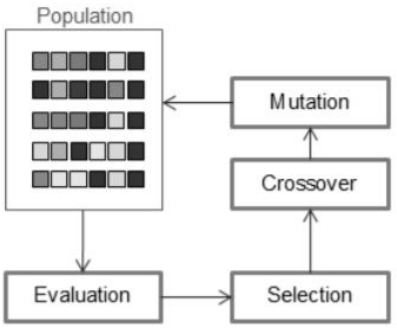

Fig 2. Illustration of genetic operators.

As it is shown in Figure 2, the GA mimics three evolutionary processes or operator: selection, chromosomal cross-linking and mutation.

A. Selection

The selection process serves to choose the individual or individuals with the highest adaptation to the environment, in other words, they are the set of the population with the highest score. In genetic algorithms, the selection is represented as a set of rules meant to *select* the parents of the next generation or iteration.

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}$$

Fig 3. Probably formula that the i individual is chosen Where N is the number of population and f_i is the fitness score.

A system widely used in genetic algorithms is the Fitness proportionate selection, also known as roulette wheel selection. This system consists of a fitness function which assigns a fitness to each individual based in their performance score. The fitness level is used to compute a probability of a given individual of been selected as shown in Figure 3. Although candidates with higher performance-score have more chances of been selected, there is still a chance that a weaker candidate is chosen. This idea is based on the fact that even though the overall performance is lower but it is not zero, which means it is still possible that in future iteration the individual may have some features which could be useful, and it could become a relevant individual in the population [7]. For this reason, we have select this selection method for our tests. *as is shown in Section VI.*

B. *Reproduction & Crossing*

Once the individuals are selected they can be reproduced. In this context, "reproduction' is the mix or cross of genes of two individuals into a new individual that will pass to the next generation. During this phase, the crossing or mix of the genes may or may not occur due it depends on a probability passed as a hyperparameter in the GA functions. The amount of genes that each parent provides is in most of the times based on a point as shown in Figure 4.

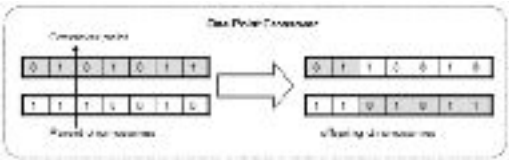

Fig. 4 Crossing based on a point.

C. *Mutation*

The mutation is a basic operator that introduces randomness into the individuals of a population and therefore variation. In the same way as the crossing rate, it is given as a hyperparameter to the Genetic Algorithm. The main goal of this operator is to change some genes of a new individual as we can appreciate in nature.

IV. Material and Methods

We carry on experiments with genetic algorithms to show how they function and prove what kind of problems they are capable of solving. For the experiments, we began from a very basic exercise up to run a robot on a simulation in order to make it crawl in an environment powered by the Unreal Engine. The evolution of the robot is provided by the Pyevolve library. It is a complete genetic algorithm framework designed for quick implementation and provides a ton of freedom to configure the hyperparameters of the genetic algorithm functions.

Fig. 5 NAO Robot crawling in the simulator.

In Figure 5 we can see the simulation of the NAO Robot. This robot has the same specifications than the real robot. It has plenty of sensors to perceive his environment and locate himself in space. Those sensors are located on the head, hands and feet, etc. Also have four microphones and speakers to interact with the humans and two cameras 2D to recognize shapes and objects [8].

V. Results

In this section, we will introduce the experiments we were asked to solve in the laboratory sections related to Genetic Algorithms. To understand the basics of this optimization artificial intelligence method we had to implement the simplest case where optimization refers to obtain a 1D array full of 0's. Following this methodology, we use it to solve the Travelling Salesperson Problem (TSP) and finally train an NAO robot to crawl.

A. *First Example*

To begin with the GA we have a simple implementation consisting of a 1D array of integers of length 10 fitted with random numbers. The objective is to convert all the numbers of the array in 0's. To achieve this goal, it is necessary to define a score-function and set the genome parameters that will be used by the GA.

A score-function is used to compute the raw score for each individual of a population. Depending on the problem meant to be solved it will be implemented differently. In this problem the score-function is simple. It goes through all the numbers of the list and returns the score by counting the amount of 0's in the array. Therefore, the best score is n: length of the list.

The genomes in the exercise represent each individual number of the array. Therefore, each individual has a different distribution of values that will be changed over each iteration. As we mentioned in Section III those changes are mainly powered by a crossing process and mutation. A set of genomes creates a chromosome, so they are the building blocks of an individual.

The Genetic Algorithm is created in conjunction with the functions defined above: score and genomes. And it needs to be fitted with the

hyperparameters of the selector method, number of generation and mutation rate. For simplicity, all of those parameters were set in default, resulting in 100 evolutions or generation with a population of 80 individuals per iteration, the mutation rate of 2% and a crossover rate of 80%.

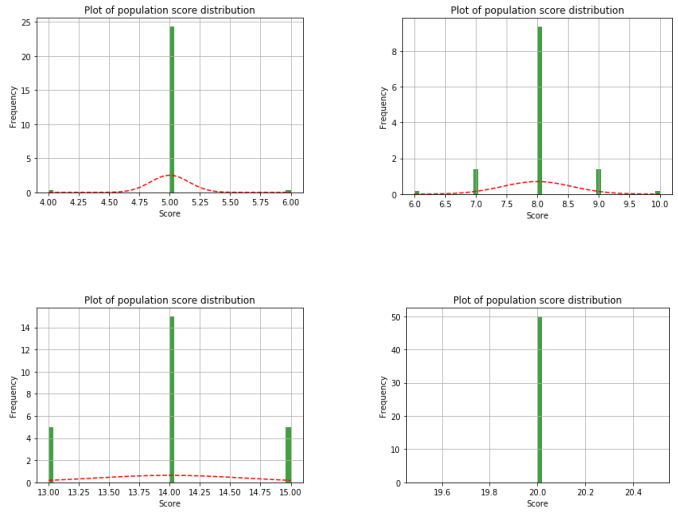

Fig. 6 Histogram plots illustrating the evolution by getting a better score in the average of the population in differents generations.

By computing the evolution step by step we were able to appreciate how each generation was performing and print the output of its best individual. The first iteration is based on a copy of the array X given as an input. Through different iterations, the score began to improve arriving at the maximum at generation 19th with an array full of 0's. This exercise worked as a proof of concept and lead to the possibility of keeping trying more complex optimization problems.

B. Travelling Salesperson Problem (TSP)

It is an optimization problem in which the goal is to obtain the most efficient way to travel through a list of different cities given the distances between each pair of cities.

First, it is necessary to create the arrays of cities and distances in order to work with them. They are the main arguments of the problem and to obtain them we used a random function. The cities are represented in the array as a 2D coordinate composed of an 'x' and 'y' component. Last, the distances are stored in a 1D array. In both of the cases, the values are obtained randomly.

For this specific problem, we developed a score function based on the total distance computed as the sum of travelling throw the set of cities in the order specified by the individual. The genome settings refer to how a certain individual will come up with the order of the cities that the Salesperson will travel through.

In the configuration of the GA, we define the mutation rate in 0.02 and crossover rate in 1.0. Finally is necessary to define the population, which we set in 80 individuals, along with the number of generations with 2000 iterations.

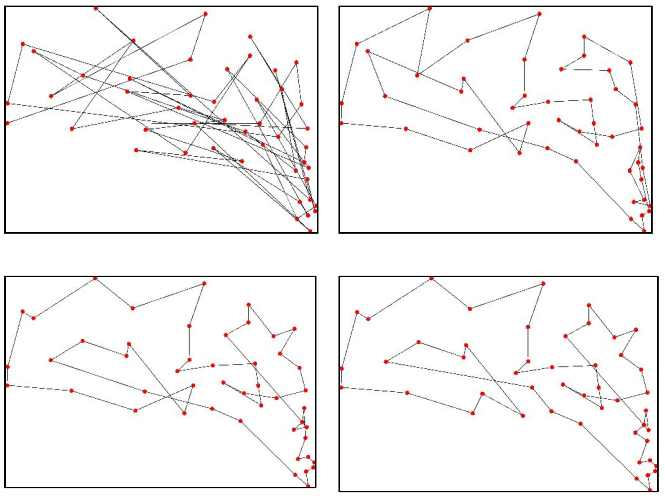

Fig. 7 The evolution of the travels, from the worst (left top image) to the best (right bottom image) travel.

By generating a plot every 100 generations we can see the evolution of the genetic algorithm. As shown in Figure 5, the top left plot refers to the first generation, we can appreciate an almost random net of connections with an average score. The top right and bottom left plot are the 500th and 1300th. Finally, the last plot refers to the last generation with the higher score, therefore with the lowest distance.

C. NAO Robot

Nao is an interactive humanoid robot of 58 centimeters and fully programmable. The robot is capable of interact with the environment around him using multiple sensors such as two cameras, four microphones, nine touch sensors, two ultrasonic sensor, an accelerometer, gyroscopes and the set of engines that work as joints in a real human body and allow him to listen, walk, crawl, etc. In addition to all of this hardware, NAO includes a graphical software that allows users with low programming skills to work with it, although for more advanced users it's possible to program the robot with proper languages like Python, Java or C++.

For experimental purposes, the NAO has been fitted in a simulation powered by Unreal Engine 4. As we mentioned before, all the parameters of the robot can be controlled and tweaked with his own library. The main goal of this project was to implement a Genetic algorithm to teach the NAO to crawl by finding the optimal parameters. For this purpose, we used Python as the main programing language to develop these experiments due to the availability of open-source libraries related to GA like Pyevolve. To control oscillators that work as joints in the NAO we need to use a total of 19 parameters. The set of the parameters forms the genome that the Genetic Algorithm will use to train the model and find a correct configuration yielding in a proper generation a the gait.

Following the structure of the previous exercises, a suitable fitness function is necessary in order to evaluate the performance of each individual of the population. We decided to develop the score function based on the distance traveled by the individual for 5 seconds. Therefore, the further the NAO goes, the higher the score it achieves. In addition, each parameter has a maximum and minimum valid range that can be fitted in the NAO but the GA generates the new values of the genes within a shared maximum and minimum range. For this reason, we had to define a scalar function that adapts each element to the previously mentioned valid range.

Once the score and scalar function are defined we began to define the configuration of the genomes and genetic algorithm hyperparameters. The genomes were implemented with a range generation between 1 and 0 (lately scaled to fit the necessity of each genome), Real Gaussian as integer mutator, Crossover Uniform as crossing method, and lastly the fitness function in conjunction with the scaled function as an evaluator. With this setup, the genetic algorithm can be implemented and set the hyperparameters. First, we used GRouletteWheel as the selector of the individuals, a crossing rate of 0.8 with a 0.5 mutation rate.

The training session was run over 50 generations with a population of 10 individuals per run. During this process, we notice that the crawling learning could yield critical failure where the NAO ends up laying face up and forcing us to reset the training. To solve this inconvenience, we modified the code and added a security function capable of handling this error. It checks whether the pose of the NAO is correct to continue with the training. In case the error is produced, the function will send the stand_up and initCrawling command to the NAO as shown in Figure 7.

Fig. 7 NAO recovering from the stumble

Due to the simulation of each individual was run over 5 seconds, for the training section we ended up spending around 56 minutes to complete it and extracted a database of the evolution process. Plotting the fitness score as a function of the number of generations, we observed a convergent evolutionary process with an average of 0.5 and a maximum performance of 0.8 between the 20th and 30th generations. For further analysis check the figure below.

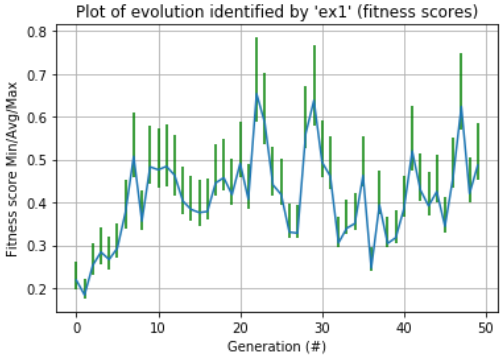

Fig. 8 Plot of the fitness score over each generation.

VI. Conclusion

The genetic algorithm is a search and optimization problem method capable of producing successful results that are basically based on the natural evolutionary process. As we discussed through the paper, GA has a wide range of uses and we explore in more detail how to implemented in three different problems. First, we started with a simple example as a proof of concept as a way to introduce the building blocks that forms a genetic algorithm by trying to convert all the numbers of an array in 0's successfully. Finally, we carried-out some test on the NAO Robot showing that it is possible to train a model capable of developing the desired feature and show the potential that genetic algorithms have.

To conclude, we would like to expose some possible improvements. The configuration of the hyperparameters, such as mutator, crossover or selector methods has big impact into the fitting-score obtained by the models so further testing may have a positive impact and they could over perform better than our trained models. In addition to the GA configuration, it requires time to test each individual during the training. To produce our testbench with 50 generations we needed around 56 minutes. By increasing the ratio generations/population, and therefore the training time could yield into a better fitness score.

References

[1] https://www.khanacademy.org/science/biology/her/evolution-and-natural-selection/a/darwin-evolution-natural-selectionn

[2] V. Mallawaarachchi, "Introduction to Genetic Algorithms", towards data science, 2017.

[3] https://en.wikipedia.org/wiki/Genetic_algorithm

[4] Sivanandam S., Deepa S. (2008) Genetic Algorithms. In: Introduction to Genetic Algorithms. Springer, Berlin, Heidelberg.

[5] https://www.britannica.com/science/natural-selection

[6] https://www.nhm.ac.uk/discover/what-is-natural-selection.html

[7] https://en.wikipedia.org/FitnessProportionateSelection

[8] https://www.softbankrobotics.com/emea/index.php/en/nao