PROJECT REPORT

MAT501 - Applied Mathematics and Artificial Intelligence

Author
Pablo González Álvarez
18 December 2023

Contents

1 Summary

The project consists of a car learning to drive using a genetic algorithm. The project has two scenes to test the efficiency and effectiveness of the algorithm. In the first scene, cars run one by one so that at the end of each lap, the track changes, demonstrating how the car adapts to different circuits. In the second scene, as many cars as the generation has neurons are executed. This is used to show how well the algorithm works based on the number of entities using it. Additionally, in this scene, the number of cars can be less than the number of neurons of the generation leaving the possibility to run a generation in more than one step.

2 Historical Context

The first genetic algorithm was created in 1954 (Carr, 2014) by Nils Barricelli. Unlike contemporary applications that focus on optimization problems, Barricelli's algorithm was originally designed to generate artificial life.

After that, the first that laid the foundations for what we now know as genetic algorithms emerged with Professor John Henry Holland in 1975 with the publication of "Adaptation in Natural and Artificial Systems" (Holland, 1992). In this work, he describes how biological evolution could be applied to optimisation problems, demonstrating the mathematics behind the evolution, within the field of artificial intelligence here he started using crossover, mutation, recombination and inversion, before him only mutation was used and it made a huge improvement. Holland's pioneering contribution laid the foundation for the application of evolutionary principles to address complex challenges in the realm of computing and optimization. From there, it began to gain widespread popularity and was studied and utilized to address complex optimization problems where conventional programming of the time and optimization methods of that era were limited. As a result, this method was turned to in order to tackle intricate optimization challenges. Currently, this method is employed in various fields such as the Internet of Things, in the economy, or to create intelligent traffic signal systems (Tanweer et al., 2023).

3 Learning technique

The genetic algorithm stands out as the optimal AI choice for this project due to its proficiency in solving optimization problems, the central challenge in this project. Furthermore, the algorithm's ability to generate diverse behaviors by tweaking just a few parameters adds to its appeal. For instance, shifting the priority from achieving the fastest lap to prioritizing the distance to the walls enables the creation of the most safest car on the circuit.

In addition to these points, genetic algorithms offer several other advantages. They excel in efficiently exploring extensive solution spaces, making them ideal for projects requiring the testing of numerous configurations to identify the optimal combination of parameters and driving strategies. Which is just what I need to get the best time on each circuit.

Genetic algorithms are particularly effective at navigating non-linear search spaces, common in complex optimization problems. The inherent complexities of racing circuits introduce non-linearity, and help discover solutions that may be challenging for other algorithms.

Furthermore, genetic algorithms have the ability to simultaneously explore multiple potential strategies within the population. This parallel exploration proves beneficial for assessing various approaches to find the most effective combination of parameters and behaviors for achieving the fastest lap.

4 Genetic Algorithm

The genetic algorithm is a method for solving both constrained and unconstrained optimisation problems based on natural selection. It involves creating an initial population, letting it work, and, based on how well each member of the population performs, either preserving or eliminating them. New members are then added to the population, starting from scratch or by mixing with the existing population. Moreover, each member of the population can mutate, except if you are in the group of the best; in that case, you cannot. This process continues iteratively until the most optimal solution is obtained or until the programmer is satisfied with the result (Katoch, Chauhan, and Kumar, 2021).

4.1 Encoding

In the project, each member of the population is a car, specifically a neural network, which is a way to process data in a computational model. Each car's neural network has an input layer with five values ranging from 0 to 1. These values represent sensors for detecting the distance of the car from the wall, looking straight ahead and at 15 and 30 degrees to the left and right. The number of sensors is arbitrary, and the more sensors added in the right direction, the better it will function. However, it will lose efficiency as it has to perform more distance calculations, so I have found 5 to be a good number, although it can also work with 3. After sending this input layer, it is translated into an output of two values: a rotation for turning the car and a speed that the car should maintain (see Figure 1).

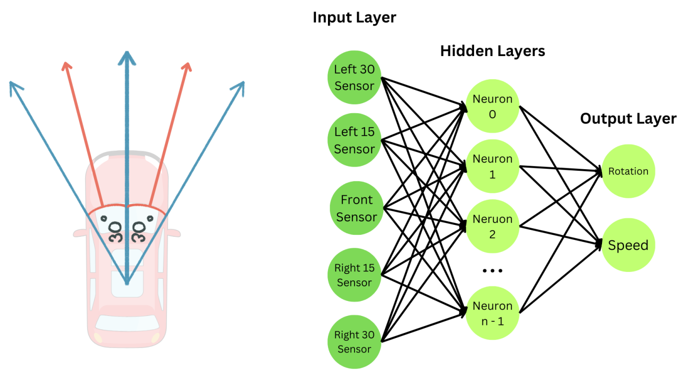


Figure 1: Car neural network

The code to create the initial population has a computational cost of O(n * m), where n is the population size and m is the number of neurons in the neural network.

4.1.1 Neural Network Mathematical Model

Once we know what parameters to pass to the neural network and what parameters it should receive, we need to calculate how to arrive at those final parameters that we are seeking (Dasaradh, 2020).

In the first step (1), we have to make the dot of the vector x, which is the input value vector, by the weights vector w; this vector represents the importance of the connections in the neural network.

$$x.y = (x_0 + w_1) + (x_1 + w_1) + ... + (x_i + w_i) \tag{1}$$

Once we have that calculation, we need to add the value b (2), which represents the bias acting as an offset and is necessary to shift the function to the left or right.

$$z = x.y + b \tag{2}$$

Once we have reached this point, to obtain the output, we just need to apply a non-linear activation function. In the case of the project, I use the sigmoid function (3) for the speed (4) because its value goes from 0 to 1 and the hyperbolic tangent function (5) for the rotation (6) because its value goes from -1 (left) to 1 (right).

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3}$$

$$speed = \sigma(z) \tag{4}$$

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{5}$$

$$rotation = \tanh(z) \tag{6}$$

4.2 Selection

In the project, the selection determines whether a car continues to the next generation or not. To make this selection work, a combination of different existing methods has been followed. On one hand, there is the elitist method (Fulber-Garcia, 2023), which allows the cars with the best fitness to move directly to the next generation (7) and prevents them from being mutated. On the other hand, there is the tournament part (psil123, 2023), although the tournament is always won by cars with higher fitness, as they are sorted into a ranking (8). Moreover, a small percentage of the worst cars is also retained to introduce more diversity in the generation (9). The computational cost of this part of the algorithm has a cost of O(n * m), and it can be translated into the following formulas:

$$\text{Elitist\_Selection}(P, k) = \text{Top\_(k)}(\text{Sort\_by\_Fitness}(P)) \tag{7}$$

$$\text{Best\_Selection}(P, b-k) = \text{Top\_(b-k)}(\text{Sort\_by\_Fitness}(P)) \tag{8}$$

$$\text{Worst\_Selection}(P, w) = \text{Bottom\_(w)}(\text{Sort\_by\_Fitness}(P)) \tag{9}$$

$$Selection = \text{Elitist\_Selection} \cup \text{Best\_Selection} \cup \text{Worst\_Selection} \tag{10}$$

where:

- G is the next population that comes from the selection system.
- P is the population, shorted with the method Sort_by_Fitness.
- b is the amount of the best cars to save.
- k is the amount of the b cars to save without using them to mutations.
- w is the amount of the worst cars to save.
- $Top\_$ and $Bottom\_$ are used to differentiate between selecting the best or the worst.

4.3 Crossover

The crossover involves creating offspring from the cars stored in the previously explained selection. In this case, the project is using the Partially Matched Crossover (PMX) method (Lahjouji, Tajani, and Sabbane, 2017). This method entails taking the content of one parent, except for a segment, which will be inherited from the other parent. In the context of a neural network, two different segments are chosen—one for the weights of the neural network and another for the biases of the neural network. Here, there is a deviation from the original version of PMX as it checks for the absence of repeated values, meaning the same value cannot be present in two locations. This is omitted here because in a neural network, it is possible for two values to be the same, and it does not necessarily make the network worse. The cost of this algorithm is O(n * m * k), where n is the population size, m is the number of neurons in the neural network, and k is the number of weights in the neural network.

Initially, a uniform crossover (Katoch, Chauhan, and Kumar, 2021) had been implemented, where each bias and weight of the neural network for the offspring were randomly chosen, with a different mask to choose between parent values. However, this approach was later replaced with the PMX because, in general, PMX tends to perform better than other options (Gan et al., 2013).

4.4 Mutation

The mutation serves in genetic algorithms to introduce variety in generations. In the project, a specific number has been defined as the probability that the weights of the neural network of the car mutate. In the event that mutation is required, two different types of mutation have been implemented to compare different results:

- **Displacement mutation (DM):** In this type of mutation, the value assigned to the weight matrix of the neural network is derived from its own value (Katoch, Chauhan, and Kumar, 2021). The weight starts with its current solution and adds a random value, ensuring that it falls within the desired range (11).

  $$weight_{(p)(m)(i_1)(j_1)} = weight_{(p)(m)(i_1)(j_1)} + r \tag{11}$$

  where:

  - p is the number of the car in the population.
  - m is number of the matrix in the car p.
  - i and j are the row and column of the matrix
  - r is a random value going from -1 to 1, always making sure that weight will not be bigger than that values.

- **Simple inversion mutation (SIM):** In this type of mutation, to modify the weight matrix, two variables are needed, and their values are exchanged in the 2 random positions (12).

  $$weight_{(p)(m)(i_1)(j_1)} = weight_{(p)(m)(i_2)(j_2)} \tag{12}$$

In the final version, SIM is being used because it generally performs better and is the most commonly used method. The cost of the algorithm used is O(n * m * k), where n is the population size, m is the number of neurons in the neural network, and k is the number of weights in the neural network.

4.5 Fitness

To determine if a neural network has been good or bad, it is necessary to define the parameters that qualify a car as good or bad. In this case, the distance travelled, speed, and distance to sensors are considered to prevent the car from getting too close to the walls. The distance to the wall is measured from the centre of the car without considering its width and length. Once the car completes the circuit without crashing, we focus on achieving the fastest lap. To do this, we already have the speed, and in addition, we have two new multipliers: the fast lap, which measures the time it takes for the car to complete a lap, and the proximity to the optimal point in the curves. This latter point will not be given much importance since the car's motion simulator is not hyper-realistic, and there are many variables not taken into account when simulating the car's movement, such as brake distribution or whether the car is driving on a clean or dirty side of the track. Taking all these parameters into account, we have this fitness formula:

$$L_d = MT - L_i \tag{13}$$

$$F = T_d * M_d + As_d * M_{as} + \frac{S_d}{3} * M_s + C_d * M_c + L_d * M_l \tag{14}$$

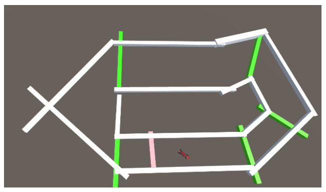
Where:

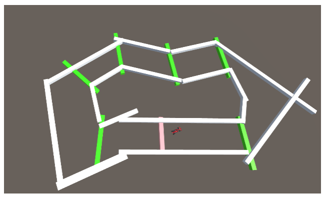
- F is the fitness value.
- MT is the max time possible to finish a lap, in the actual circuit 20 seconds.
- i is the number of lap.
- L_i is the time in seconds of the lap i.
- T_d is the total distance traveled.
- As_d is average speed in the T_d.
- S_d is the sum of all distances to the wall from the different sensors.
- C_d is the sum of all distances to best point in the corners.
- L_d is the value after the time of all laps finished, first iteration is not taken into account because it starts from a stationary position.
- M_d is the multiplayer of distance, I high value means you are giving a high importance to the distance. In the last version of the projects it is 1.4.
- M_{as} is the multiplayer of speed, I high value means you are giving a high importance to the speed of the car. In the last version of the projects it is 7.
- M_d is the multiplayer of sensors, I high value means you are giving a high importance to the distance to the walls, this could be useful if you wanna make a safe car. In the last version of the projects it is 0.2.
- M_c is the multiplayer of sensors, I high value means you want to move the car to move close to the best points in the corner. In the last version of the projects it is 4.
- M_l is the multiplayer of fast laps, I high value means you want to have fast laps. In the last version of the projects it is 7.

5 Analysis

To analyse the performance and efficiency of the algorithm, three scenarios have been devised. In the first scenario, the best time achieved by the AI is compared when training on each circuit separately, contrasting it with the time achieved when circuits are mixed (After each lap the circuit change from a pool of circuits). In the second one, it is checked how long it takes for the AI to complete the first full lap without errors to assess how easily it accomplishes this. In the last, we check how many cars can be running at the same time in the IA until the FPS make it unplayable to see how good the algorithm perform.

The tests have been conducted on the following circuits: The first (see Figure 2a) and second (see Figure 2b) circuits are very similar, with the only difference being the shape of one curve. Therefore, the results in terms of time should be similar. On the other hand, the third circuit (see Figure 2c) is slightly more complex and a bit longer. In all circuits, the red block signifies the finish line, and all tests have been conducted with the same parameters for all cars.


(a) Circuit 1


(b) Circuit 2

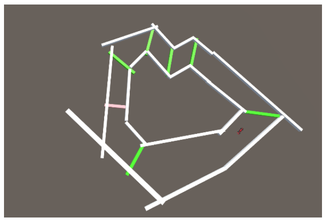

(c) Circuit 3

Figure 2: Test tracks

5.1 Performance

To study the performance of the algorithm, we conducted different experiments. On one hand, we ran the programme to see what is the actual best time he can do and on the other hand we test how many time it takes to the algorithm to finish one lap.

5.1.1 Best Time Experiment

In the first part, we got this first graph (see Figure 3). The graph illustrates that, right from the start, the algorithm found a quite accurate solution for the fastest lap on circuit 2, whereas it took more time for circuits 1 and 3. The abrupt changes between points are likely attributed to a mutation, as improvements in lap time generally occurred from one car to another rather than gradually. Conversely, when there is only a slight improvement in time, it is probably due to the creation of offspring between two parents with high fitness.

In the second part, where all circuits are tested simultaneously, it can be observed (see Figure 4) that the results at the start are much worse and at the end are slightly worse. This could be attributed to the fact that each car, until it crosses the finish line, is unaware of the next circuit. Consequently, when the circuit changes upon crossing the finish line, its perception of distances may change, affecting the algorithm's performance.

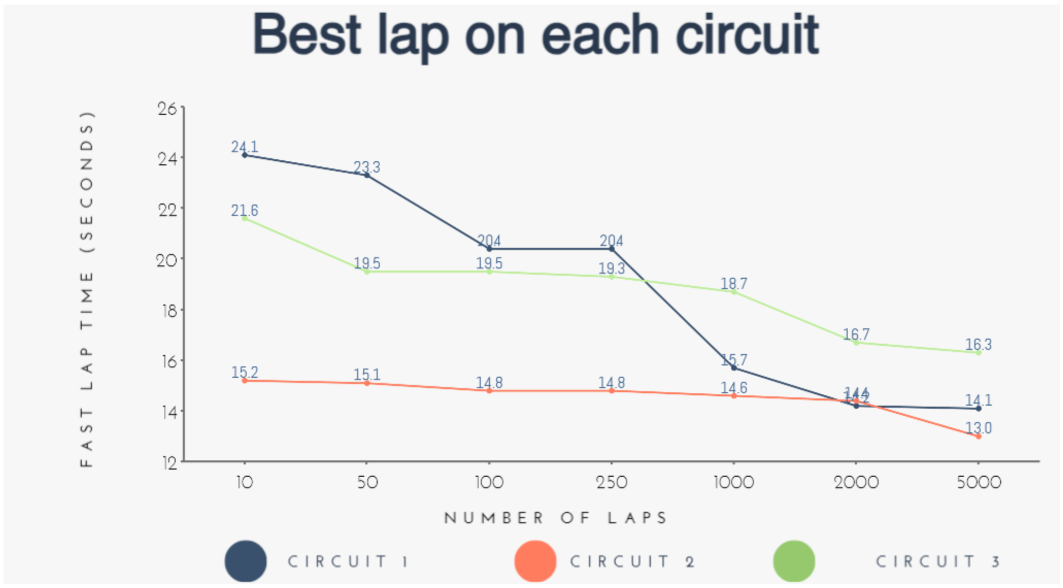


Figure 3: Separated Tracks Test

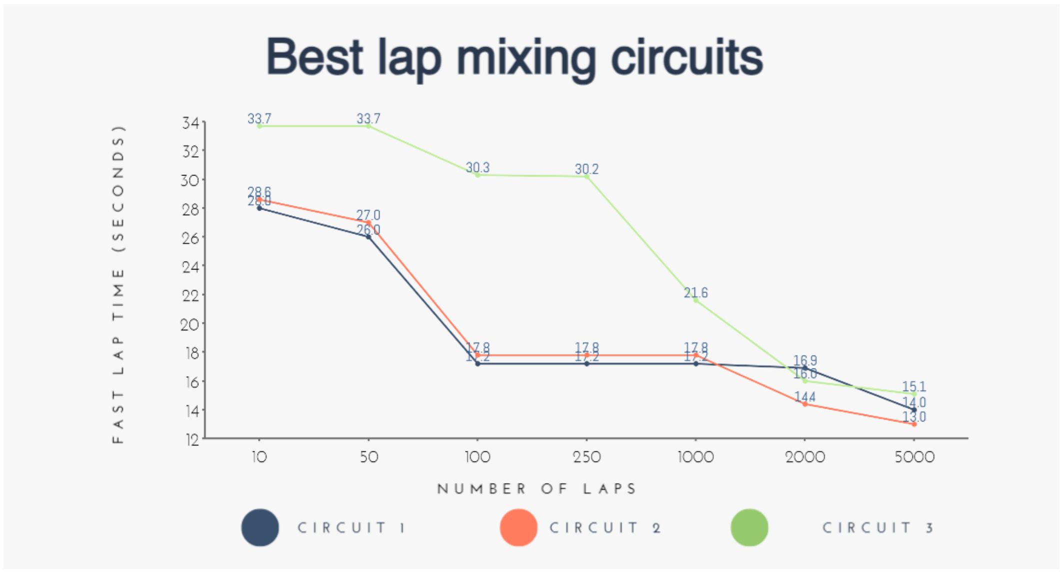


Figure 4: One Lap Test

5.1.2 Mixed Tracks Test

In this experiment, as shown in the image (see Figure 5), it can be observed that as curves are added to the tracks, the time it takes to find the solution increases. Although the final lap times are similar, the third circuit takes significantly longer than the others due to the presence of tight curves at the beginning, even though they can be navigated almost in a straight line.

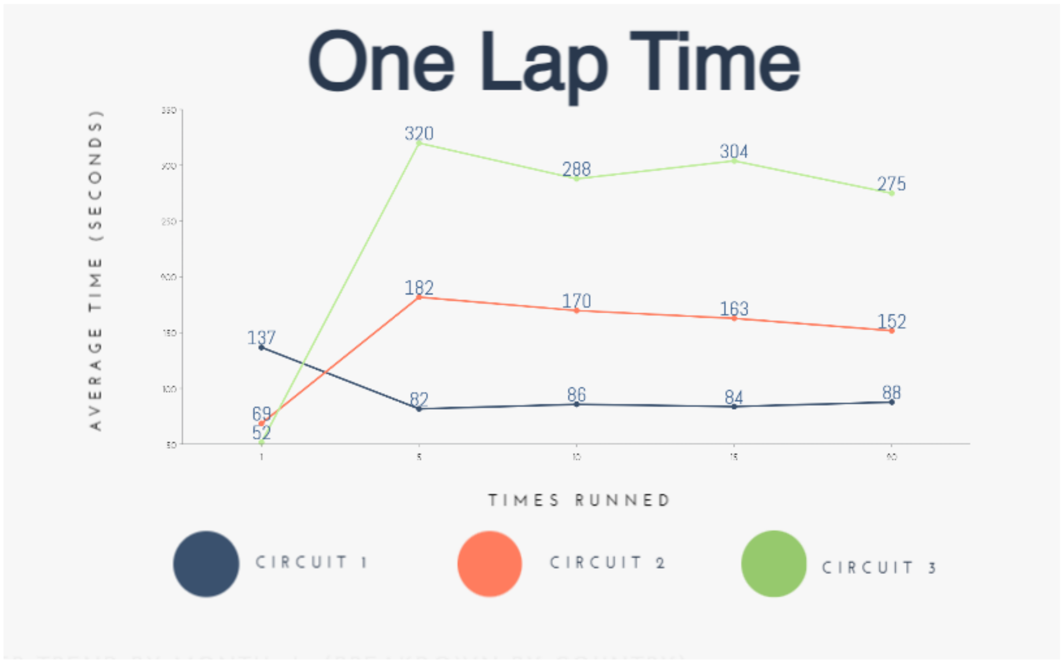


Figure 5: One Lap Test

5.2 Efficiency

To study the efficiency of the algorithm, on one hand, we will investigate the limit of possible cars before the game becomes unplayable. On the other hand, during the explanation of the algorithm, the computational cost of each one was explained.

Looking into the number of cars the application can handle, we can see in the graph (see Figure 6) that the algorithm performs well up to 25 simultaneous cars. However, beyond 100 cars, it becomes impractical to generate more cars due to the low frame rate obtained. Even though running with many cars simultaneously may not be feasible, it is possible to have large generations by allowing fewer cars to run and having them run multiple times in the same generation but with different neural networks.

Note: The performance tests were conducted on an MSI GL65 9SDK laptop with the following specifications:

- Processor: Intel Core i7-9750H
- Graphics: NVIDIA GeForce GTX 1660 Ti
- RAM: 16GB DDR4
- Storage: 512GB NVMe SSD

6 Conclusion

In conclusion, the project fulfils its intended purpose, although there are various aspects that could be improved. I believe the efficiency issue arises because the project is in 3D, and consequently, collision calculations are also in 3D. This could have been avoided by

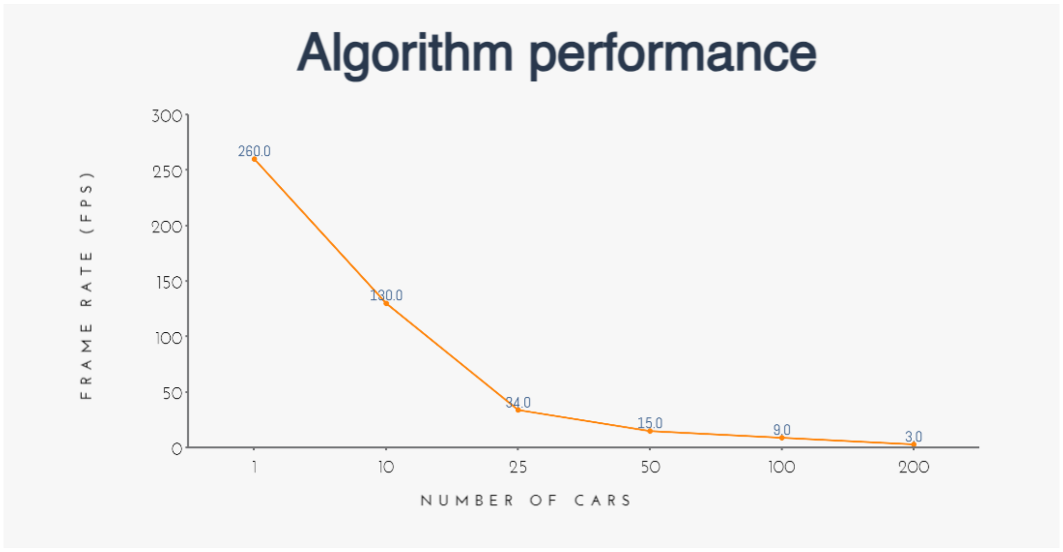


Figure 6: Performance Test

implementing the project in 2D, yielding similar functionality. Despite the observed improvements in the analysis of lap times, I think there is room for further enhancement by playing with all the adjustable parameters. The chosen configuration is what has yielded the best results based on my experimentation.

Lastly, the car's movement itself is quite simple, and to create a simulator that closely emulates real-world movement, additional parameters would need to be incorporated for both the car and the track, such as brake distribution, tyre type, asphalt type, etc. Implementing these additions would require a significant amount of time and expertise in mechanical engineering. Nonetheless, the project serves as a solid starting point for incorporating such changes.

References

Carr, J. (2014). "Computational Intelligence in Automotive Applications". In: *https://www.whitman.edu /documents/academics/mathematics/2014/carrjk.pdf*. Accessed: 11 December 2023.

Dasaradh, K (Oct. 2020). "A Gentle Introduction To Math Behind Neural Networks". In: *https://towardsdatascience.com/introduction-to-math-behind-neural-networks-e8b60dbbdeba*. Accessed: 9 December 2023.

Fulber-Garcia, V. (June 2023). "Elitism in Evolutionary". In: *https://www.baeldung.com/cs/elitism-in-evolutionary-algorithms*. Accessed: 7 December 2023.

Gan, S. et al. (Dec. 2013). "A Comparison on the Performance of Crossover Techniques in Video Game". In: *https://www.researchgate.net/publication/261206130_A_comparison_on_the_performance_of$_c$rossover_techniques_in_video$_g$ame*. Accessed: 7 December 2023.

Holland, Jhon H. (1992). *Adaptation in Natural and Artificial Systems*. The MIT Press.

Katoch, S., S.S. Chauhan, and V. Kumar (2021). "A review on genetic algorithm: past, present, and future". In: *https://link.springer.com/article/10.1007/s11042-020-10139-6Sec1*. Accessed: 6 December 2023.

Lahjouji, A., C. Tajani, and M. Sabbane (May 2017). "New crossover operator for genetic algorithm to resolve the fixed charge transportation problem". In: *https://www.researchgate.net/publication/ 316798704_New_crossover_operator_for_genetic_algorithm_to_resolve_the_fixed_charge_transportation_problem*. Accessed: 7 December 2023.

psil123 (2023). "Tournament Selection (GA)". In: *https://www.geeksforgeeks.org/tournament-selection-ga/*. Accessed: 7 December 2023.

Tanweer, A. et al. (June 2023). "Genetic Algorithm: Reviews, Implementations, and Applications". In: *https://arxiv.org/abs/2007.12673*. Accessed: 7 December 2023.