Path Optimization in Stochastic Environments using Genetic Algorithms

Espen Peterson
Rochester Institute of Technology
Kate Gleason College of Engineering
Rochester NY, USA
esp8704@g.rit.edu

Anthony Ambrose
Rochester Institute of Technology
Kate Gleason College of Engineering
Rochester NY, USA
axa6841@rit.edu

I. Problem Statement

The goal of this project is to develop a genetic algorithm that can determine a safe path across a stochastic and dynamic environment. The purpose of this program is to model how a real population might go about solving a the dynamic problem of crossing a busy road. The end result should be a group of agents that are able to determine a safe path across a simulated busy road with a rewards structure and with no constraints on their movement except for not being able to revisit the last visited tile.

Although there have been other studies that utilize genetic algorithms and reinforcement learning for solving mazes, there have been none that solve a problem with some form of dynamic element, such as the moving car in our problem. Due to the moving car and the pygame element each iteration cannot proceed til the visual simulation is complete. This aspect of the project makes it a lot slower than a normal algorithm but provides a real time, visual demonstration to the user.

This paper will further establish this project's claim on unexplored areas with genetic algorithms with a brief literature review section. The methods employed in this study and the reasoning behind them will be explained. Following that, the validation methods will be discussed.

II. Literature Review

Genetic algorithms are inspired from biological systems and mimic how nature solves problems and optimizes in an iterative way. Specifically, there are also genetic algorithms that have been developed to solve mazes and find shortest paths. One paper [1] describes a genetic algorithm that uses a fitness switching function to determine the best path through a maze where most paths are not feasible. What will differentiate this proposed algorithm from that is that one the task environment will not be deterministic, and two there will be elements of danger in the task environment. This means that the agents will need to reevaluate their fitness score based on a random occurrence.

Another paper [2] is focused on comparing genetic algorithms to path finding algorithms like A*. Its application, focuses on finding a less computationally expensive way to implement path-finding in games. This definitely lends some credence to the idea that genetic algorithms may be able to outperform search algorithms. This project has a lot of parallels to both our methodology and task environment, however the differences are also numerous. While [2] is dealing with an environment that has no danger or need for readjustment, this project has that as a core component. Additionally, the goals outlined in [2] are almost entirely oriented around outperforming this search algorithm, while ours are to see if the genetic algorithm will work for a dynamic environment at all.

Another approach for solving randomly generated mazes is discussed in Nitin Choubey's paper [5]. These mazes, although randomly generated, do not have any dynamic elements in them, however. In other words, once the maze is generated, the GA is left in a unchanging environment to solve it.

Moving slightly away from academics, a blogger tried his hand at genetic algorithms [3] and documented the project. He had a few maze layouts and showcases the implementation of his genetic algorithm. His was a beginner approach and his background was not in genetic algorithms. However, viewing a beginners approach is extremely helpful because potential pitfalls are shown as well as examples of inefficient or inadequate implementation. In this specific case, the author gives areas of the project that could have been improved and specific resources to use. The first maze was a normal maze that the algorithm was able to solve in about 5 minutes. The final maze was a "maze that tortures the GA", because it goes from each side of the screen to the other side, in a snake like pattern. Due to the nature of genetic algorithms, it was unable to solve the final maze because the simulation terminated at 50 minutes without ever hitting its goal, although it would have given enough time.

A paper published in 2004 [6] details a path finding approach in which the path is evaluated according to the length, smoothness, and clearance. Each node has these qualities and an evaluation of each node provides a cost map. [6] helped provide insight into potential methods that could be used in this study. One such insight was the method of chromosome data structure, shown below. The chromosome represents the path as a sequence of nodes, split into their x and y components. This method was adopted into this project as well.

Mostapha Kalami Heris created the Yarpiz Project [7], which aims to be a resource for source codes and tutorials, specifically targeting artificial intelligence. There were several "Genetic Algorithm in Python" videos that helped us set up the skeleton of our algorithm. In addition to the tutorial videos, he also has a Python package called "ypstruct" [7], which is a simple and easy to use MATLAB-like structure that is utilized in this project. They provide all the functionality of a class/object in Python, but a little simpler to implement.

III. Methods

A. Task Environment

The task environment was created using pygame. It was comprised of a grid of squares containing different squares and having different costs associated with them. The entire map was 17 tiles wide and 12 tiles tall. The objective was to teach the turtles to safely cross a bridge set up over a road and reach the right edge of the map. The map remains the same from epoch to epoch, but it is randomly generated at the start of an experiment. The bridge placement is also random, with the one constraint being that it spans the road.

The dynamic portion of this environment is the cars, which travel across the screen every 2 seconds. An important feature of the bridge is that being on the bridge provides the turtle with immunity from the cars. The following figure shows the task environment as a new generation of turtles spawns on the left side of the map and starts to make their way across.

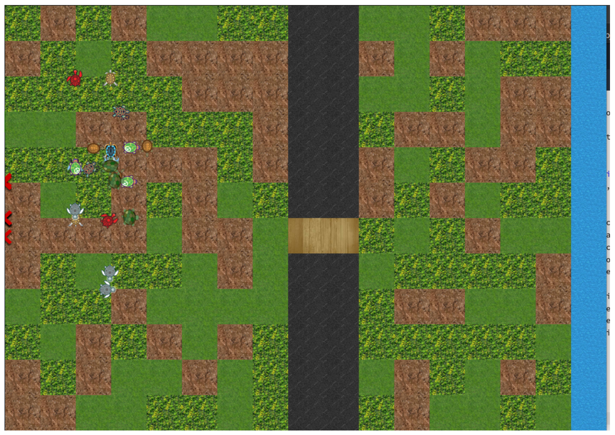


Fig. 1. Task environment

The turtles in the environment are each their own independent object. They have several key attributes worth discussing. The `path` variable is a list of coordinates that the turtle will travel to. This variable starts out as a randomly generated series of movements, and for each epoch after the first is generated from the `gene` attribute from that turtle. The `gene` element describes the same movement pattern as the `path` element, but it is in terms of cardinal and sub-cardinal directions instead of coordinates.

Each turtle also has flags that are set to true if it comes into contact with other objects in the environment, such as contact with the bridge, a car, or water. These flags are later used to help determine the cost of the turtle. The final x position of the turtle, as well as the amount of movements it had to do to get there also count towards the turtle's cost. The biggest factors in determining cost are whether or not the turtle gets hit by a car, touches the bridge, and reaches water. Getting hit by a car results in an extremely high cost for the turtle, which pushes that path out of the gene pool. Conversely, a turtle reaching the bridge and water is rewarded. However, turtles that reach the water without using the bridge are not rewarded due to the fact that crossing the road in places other than the bridge has a random chance of death and is therefore an undesirable path.

B. Methodology

A detailed flowchart is shown in Figure 2. The code starts off with the turtles being initialized according to the input parameters and the map being randomly generated. Then, the simulation begins and main.py runs the game. Once the game iteration ends the data gets sent to the genetic algorithm where the genes are spliced and mutated. A children turtle set is created with the new genes and if it is not the final iteration, the simulation runs over again with the new turtle set. In the final iteration, the average cost over iterations is plotted using `matplotlib` and the simulation ends.

Doing this for objects with a pre-determined path was not simple. The turtle's genes were stored as a list of one of the 8 cardinal and sub-cardinal directions. Splitting and mixing these genes worked for generating variance among the top performers, but it did little in the way of improvement. This is because putting together two parts of two different paths does not necessarily guarantee a better, or even a valid path.

Mutations serve to further vary the children from their parents, to give the child generation further capabilities. The mutation factor, `mu`. The mutation factor was set to be the inverse of what it mathematically implies in this program. `mu` was set to 4 in the experiment shown in this paper, meaning that every 4th element in each gene was mutated. Therefore a higher mutation factor will yield a less mutated gene. A visualization of the genetic algorithm's crossover and mutate functions is shown in Figure 3.

The third and most important to this genetic algorithm was the elite turtles. The number of elites was set to 3 in this experiment, which helped speed up the convergence of the algorithm. The elite turtles were immune to the splitting and the mutating, and subject only to the sorting. This meant that there were no genetic changes imposed on the 3 best performers from each epoch. They would be filtered out only if they were no longer in the top n turtles, where n is the population size.

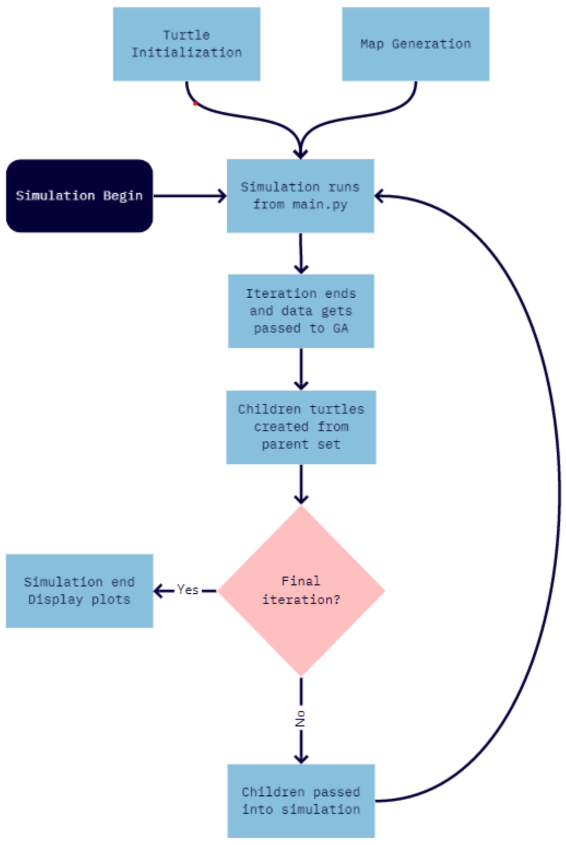
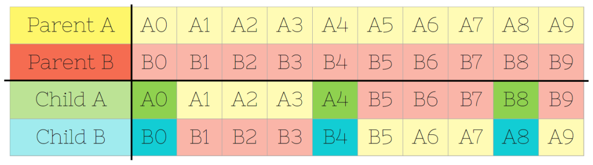

Turtle Initialization
Map Generation
Simulation Begin
Simulation runs from main.py
Iteration ends and data gets passed to GA
Children turtles created from parent set
Final iteration?
Yes
Simulation end Display plots
No
Children passed into simulation

Fig. 2. Gene Editing



Parent A	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9
Parent B	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9
Child A	A0	A1	A2	A3	A4	B5	B6	B7	B8	B9
Child B	B0	B1	B2	B3	B4	B5	A6	A7	A8	A9

Fig. 3. Gene Editing

C. Validation

Some of the parameters that are to be investigated are mutation rate, population size, iterations, and the number of elites per iteration. The cost, crossover, and mutate functions can also be changed and the impact observed. However, it is a lot easier to change set parameters like population size than it is to change entire functions. Although this maze can be solved using a better search algorithm like uniform cost or Greedy search, a genetic algorithm is unique in its iterative approach and exploring the capabilities of genetic algorithms is a goal of the project.

There will also be information on the generation being tested that is plotted over the course of the experiment. The average cost over iterations is plotted using `matplotlib`.

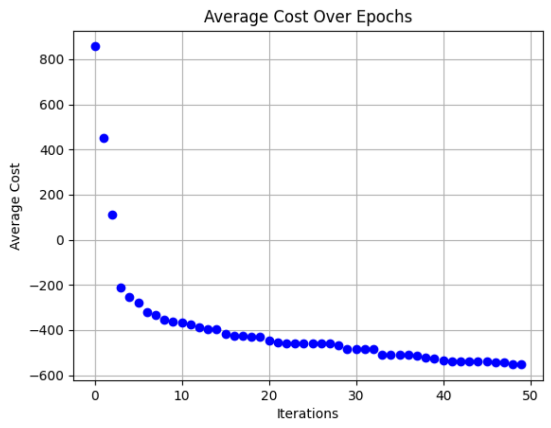


Fig. 4. Results of 50 Epochs

This paper's hypothesis is that the GA will work eventually, but will take several iterations to do so. Of course, this can be fine-tuned using the aforementioned hyper-parameters. The genetic algorithm simply working is not as interesting as the gap between a GA's problem solving capabilities and the capabilities of a normal search algorithm. Beyond getting all the turtles across the bridge and to the water, one should expect to see the GA optimize the route as iterations progress.

IV. Results

Over the epochs, the average cost in Figure 4 can be seen to consistently go down. The best cost changes very little due to the inclusion of elites, however. Although this would ideally improve steadily along with average cost, a solution that resulted in this was not found, as causing move variety and mutation with the turtles often lead to a much higher average cost.

Besides the graph, there is the game. It should be mentioned that the main reason for making this into a game rather than a piece of software that performs rapid computation for path finding was so that the user could have the opportunity to watch the genetic algorithm improve the population in real time. Even between the very first epoch, where the turtles pretty much all fail immediately, and the second one is almost always displays noticeable improvement.

After ten or so epochs, the turtles actually start grouping together much more as they head towards the bridge. Due to this, the final population size was chosen to be 30, with a total of 50 iterations. In hindsight, the main value of this experiment was the visual aspect, which provided a unique and accessible method for validation.

V. Conclusion

A takeaway from this project is that genetic algorithm is a search algorithm that works best when the solution space is full of viable solutions and parameters. Any kind of well defined NP problem, such as path finding, is not an efficient use of a genetic algorithm. Well known path-finding algorithms such as Dijkstra's algorithm, A* search, or Greedy search would be able to solve this maze in seconds. The limitations of the project was the nature of genetic algorithms and the nature of the NP problem that it was given to solve.

One particular avenue that was explored was disallowing immediate backtracking in the child turtles' paths. However, bringing in domain knowledge would also reveal other inefficiencies in the code. For example, the maze could be preprocessed into a series of possible routes and a cost analysis could be done on each path, then the algorithm chooses the best one. This would turn the genetic algorithm into a regular algorithm. Due to this, it was decided to leave as much as possible to the genetic algorithm in order to keep it a genetic algorithm.

A future implementation would either adjust the task environment to provide a more suitable problem for a genetic algorithm to solve. An example of a better task environment would be any simulation representative of an ecosystem. In addition, a genetic algorithm framework can be used in order to create a more robust algorithm.

Our hypothesis was proven right in that the genetic algorithm was able to solve the stochastic maze and reliably get the turtles to the water. However, we did not anticipate the genetic algorithm being so inefficient compared to other searching algorithms.

In terms of learning about genetic algorithms, this project was very illuminating. It was not at all what was expected going into the project; however, genetic algorithms can be implemented in intriguing ways that will be fun to explore in the future. Although it would require much more robust, real world data, it is highly likely that genetic algorithms could be used to simulate and predict many different biological scenarios.

REFERENCES

[1] Kim, JunWoo, and Soo Kyun Kim. "Genetic Algorithms for Solving Shortest Path Problem in Maze-Type Network with Precedence Constraints." Springer, Springer, 12 Apr. 2018, link.springer.com/article/10.1007/s11277-018-5740-3.

[2] Leigh, Ryan, et al. "Using a Genetic Algorithm to Explore A*-like Pathfinding Algorithms." IEEExplore, IEEE, 4 June 2007, ieeexplore.ieee.org/abstract/document/4219026.

[3] G. (2017, January 11). Path Finding with Genetic Algorithms [Web log post]. Retrieved September 22, 2020, from https://yoloprogamming.com/post/2017/01/11/path-finding-with-genetic-algorithms

[4] Ji, Z., Kim, Y. and Chen, A., 2020. Multi-Objective -Reliable Path Finding In Stochastic Networks With Correlated Link Costs: A Simulation-Based Multi-Objective Genetic Algorithm Approach (SMOGA). [online] ScienceDirect. Available at: https://www.sciencedirect.com/science/article/abs/pii/S0957417410006925?via%3Dihub

[5] Choubey, Nitin S. A-Mazer with Genetic Algorithm. Researchgate, Nov. 2012, https://www.researchgate.net/publication/233786685_A-Mazer_with_Genetic_Algorithm

[6] Elshamli, A., Abdullah, H., amp; Areibi, S. (2004). Genetic algorithm for dynamic path planning. Canadian Conference on Electrical and Computer Engineering 2004 (IEEE Cat. No.04CH37513). doi:10.1109/ccece.2004.1345203

[7] Heris, Mostapha Kalami. "Genetic Algorithm in Python - Part A - Practical Genetic Algorithms Series." YouTube, YouTube, 8 Jan. 2020, www.youtube.com/watch?v=PhJgktRB1AM.