Evolution Algorithm Application in Path Planning

CISC 455 Final Project

Shouyue Hu
20095119

1. Abstract

This report solves path planning problems via the Evolution Algorithm, testing on the map of flat ground, real-world, and maze. Then, the Breadth-First Search algorithm is compared with EA regarding the runtime of path discovery.

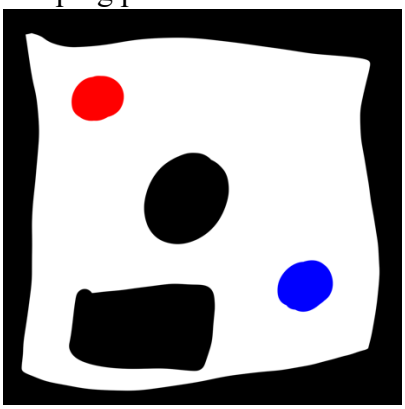
2. Introduction

Path planning has been a common problem in considerable scenes in life, such as navigation apps, robot controlling and automatic pathfinding of game agents. The scenarios of application of path planning vary a lot; would there be an effective algorithm? There are two candidates, one is classical methods including Depth First Search, Greedy Search and A* algorithm, etc. and the other methods are more dynamic and complex in implementation, such as Reinforcement Learning, Deep Learning, and Evolutionary Algorithm (EA). Evolutionary computing has been recognized performing well in global optimization and static problem solver; therefore, this report will explore how Evolution Algorithm works in the path planning problem. To implement that, rules need to be designed to specify mutation, crossover, parent selection, survival selection and population parameters. And the performance will be examined compared to alternative algorithms.

3. Methods

3.1 World reconstruction

Before path planning, robots should understand the world. There should be an environment reconstruction process that notices the robot what is walls and where are the starting location and destination are. Intelligent vehicles and drones usually employ a Simultaneous Localization and Mapping (SLAM) system via cameras or sensors and generate environment data that looks similar to these two images (Figures 1 and 2). Black is the wall and obstacle that the robot cannot pass through, white is the ground that the robot is able to move on, the red point represents the departure position, and the blue points indicate the goal. In particular, (Figure 1) is a map simulating the case in real-world that “How to reach the goal and circumvent all obstacles?”; (Figure 2) is more like a maze and the robot is the maze solver which is required to find an escaping path.



(Figure 1)

(Figure 2)

The real world is continuous, and the position space is too large to compute; as a result, a fitting grid world will be generated to reduce complexity. It can be witnessed that each colour has to be assigned a digital representation (0 for wall, 1 for ground, S for Start state, D for destination). Now the problem is changed to finding a path that starts at "S", passes through "1", and finally reaches "D" (See Figure 3 and 4). We can transform the image map into a 2-D list via the OpenCV-Python package.

(Figure 3)

(Figure 4)

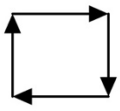
3.2 Representations

Genotype and Phenotype are the basis of EA.

Genotype is the subject of evolution so that it should be simple and modifiable. A list of integers is utilized where 0 indicates "Go North", 1 indicates "South", 2 indicates "West", and "3" indicates "East". Phenotype is the genetic expression of Genotype, and in this problem is a path initiating from starting location.

For example,

[3, 3, 3, 3, 3] represents going East by 5 steps.



[0, 3, 1, 2] represents a clockwise rotation.

3.3 Mutation

Mutation includes appending and deletion. With the probability of 0.5, append-mutation appends move with a random direction at a random index of the path. With the probability of 0.5, delete-mutation deletes a move at a random index of the path. Both mutations create new traits in the population.

3.4 Crossover

One-point crossover is applied. The reason why maintaining the complete order is that in path problems, the sequence is essential. The crossover that breaks the absolute order namely partial order crossover does not make any sense to the path. In most cases, a path walks along the wall, a destruction to order may let the path hit the wall (Figure 6) or stuck in a dead conner which will destroy fitness. Also, one point crossover persists the continuity of the path, allowing it to go straight instead of soundless rotation.

(Firgure 5)

non-order crossover (Frigure 6)

3.5 Fitness function

The idea of Manhattan Distance is used which is the expected minimum distance between the current position and the destination. For example, from [0, 0] to [3, 4] would go through Manhattan Distance of 7 = (3-0) + (4-0). It is the lower bound of potential steps to reach the goal and only happens in the best case that no obstacle blocks the path. Plus, Manhattan Distance is a greedy algorithm because the real distance only equals Manhattan Distance in the optimal case.

In this problem, the fitness of a position is the negativity of absolute distance to goal position:

$$fitness = -1 \times MhtDis = -\sqrt{(x - x')^2 + (y - y')^2}$$

Also, reaching the goal state will be reward immediately *10000 pts*.

However, a problem happens that, the robot may be easily tracked in a local optimal of a dead conner of maze because any affords of escaping the local maximum will bring temporal reduction to fitness and make it less competitive in the Parent selection. The problem will be solved in the *5. Discussion* section by adding the ε-greedy parameter and modification to mutations.

3.6 Parent selection

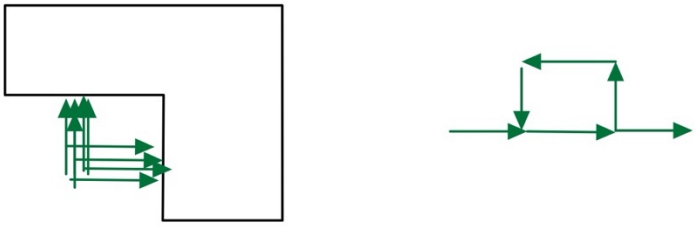
Two mating pools will be randomly selected, and one tournament will be held for each pool regarding an individual's fitness. The winners take the chance to generate offspring.

3.7 Survival selection

With the probability of 0.33, the two off-springs of tournament winners replace the loser individuals. With the probability of 0.66, the tournament is held uniform-randomly without considering the fitness, in other words, each individual takes an equal chance to win or lose the tournament.

3.8 Remove redundancy

Two kinds of redundancy are supposed to be cut off. One such redundancy is an illegal move, particularly hitting the wall. For instance, going into a wall will remain the robot at the same location but consequently increase path length without any soundness (Figure 7-left). The other redundancy is the loops. By drawing a circle, the robot walks back to the initial location with some meaningless moves (Figure 7-right). Removing the redundancy will decrease computation cost and let more mutations be conducted to the key path instead of redundancy and therefore boost the problem-solving process.

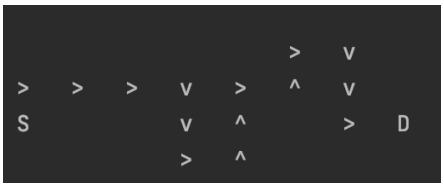


(Figure 7)

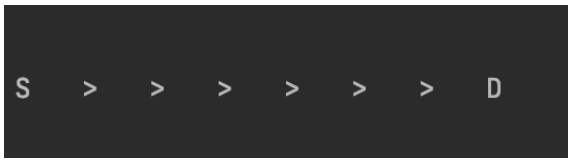
4. Results

4.1 Flat ground

First, create a flat ground without obstacles. The solutions found are figures below. By tracing the generation count, the first solution appears at the at generations averagely around 300 with a population size of 500 (Figure 8). By statistics, the quickest solution was captured at generation 3 when the initial randomly generated path is coincidently fit. But if continuing to run the iteration, a shorter path will be discovered (Figure 9).



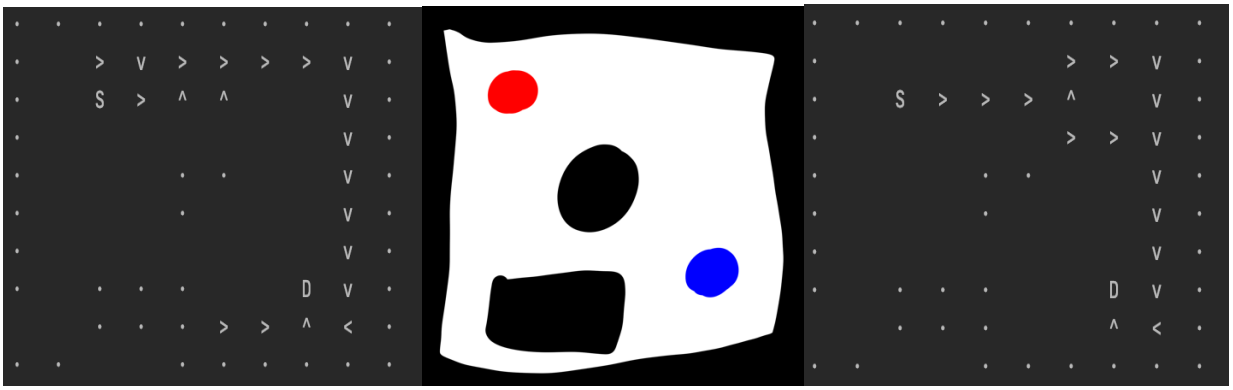
(Figures 8)



(Figures 9)

4.2 Map case 1

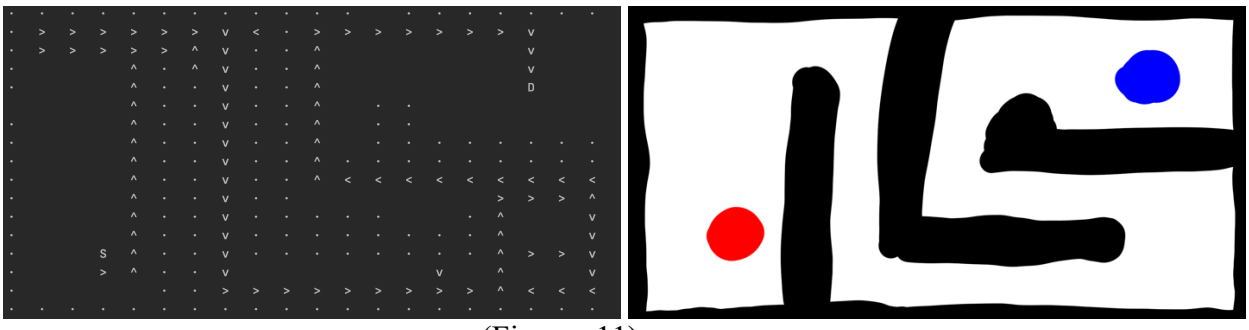
The first solution appears at the at generations averagely around 300 with population size of 500



(Figures 10)

4.3 Map case 2

The first solution appears at the at generations averagely around 3200-4200 with population size of 500 (Figure 11).



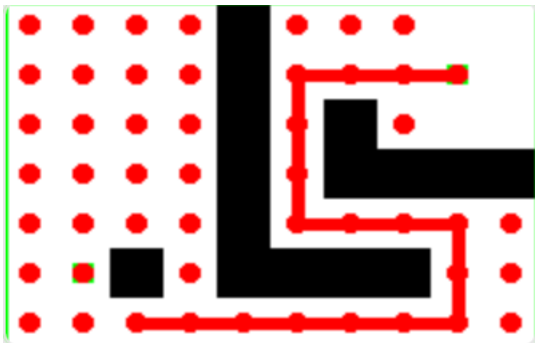
(Figures 11)

4.4 Alternative algorithms

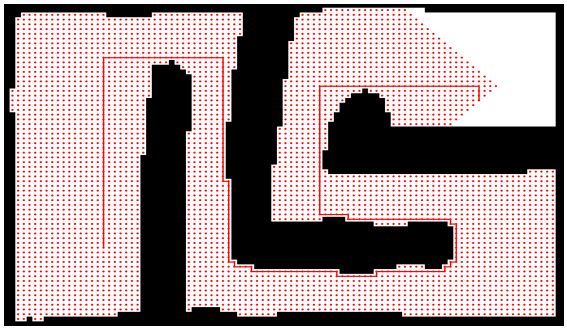
Breadth-First Search algorithm [1], as a common maze algorithm is tested in the same map. BFS generates path splits to all possible directions and will stop once a path is firstly found so that the path is shortest and optimal. BFS is a brute-force algorithm, so the optimal solution is guaranteed

which is more robust than EA. Seeing that in (Figure 12), the red region is the space that has been searched, starting from starting position. And the red line indicates the shortest path.

Additionally, when the dimension of the grid-world is relatively small (7x10), BFS discovers the solution quicker than EA. Does it say that BFS performance much effective and reliable than EA? When the dimension is small, it could be the fact; however, when the dimension grows, BFS suffers from an exponential rise in the search space (Figure 13). The next section is going to distinguish in what case which algorithm runs better.



(Figure 12)



(Figure 13)

5. Discussion

Some problems exist so new features need to be designed. After that, an alternative algorithm is compared to the EA for analysis.

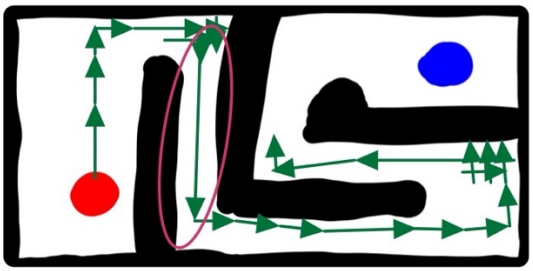
5.1 Local optimal trap

Mentioned in the 3.5 fitness function is that the Manhattan distance is a greedy scalar that asks the robot to approach the goal as close as possible. Nevertheless, going through a globally optimal path may not always decrease the Manhattan distance. So, the robot may be trapped in the dead conner for example (Figure 14).

(Figure 14)

To handle this situation, two methods are designed. Parameter ε is introduced, indicating the probability of random survival selection. That allows the robot to sometimes walks opposing the

goal and jump out of the dead conner. Plus, a stochastic step size in the pre-defined range can be added to mutation, this increases the chance of jumping out of the local maximum. For the step size range tuning, it is expected to fit the maze size. For example, (seeing a circle in Figure 15, different arrow length shows dynamic step size), the upper bound of length of step size should be slightly above to the length of dead conner in the maze, so making it easier to escape.



(Figure 15)

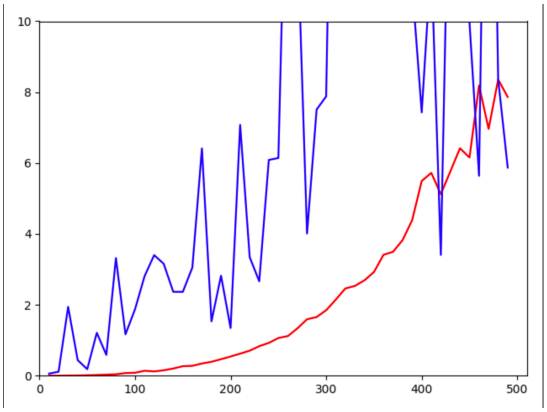
5.2 Dimension of grid world decides the complexity

(Figure 16) The X-axis is the dimension of the grid world of the first map (Figure 3) and the y-axis is the time spend to finish an algorithm in seconds.

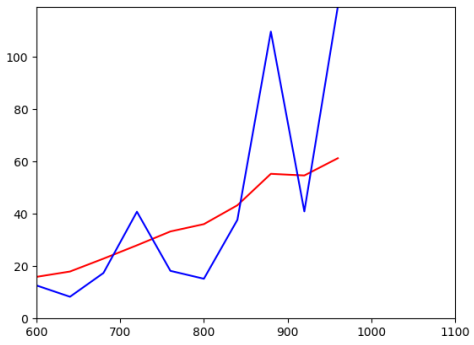
The red line shows how the runtime of BFS increases according to the growth of dimension which is in the exponential trend. By analysis, BFS search all 4 directions in every state, so the complexity is O(n^4).

The blue line suggests that the runtime of EA goes up along with dimension but not as fast as BFS. It is suggested to ignore the outlier peaks of fluctuations because that extreme long runtime only appears when a path cannot be found, and the iteration reaches our pre-defined limit. It is noticed that EA firstly runs faster than BFS when the dimension grows to approximately 420. After the historical point, EA seems to find the path faster than the BFS more frequently.

(Figure 17) For dimensions 600-1000, BFS and EA performance is similarly with fluctuations.

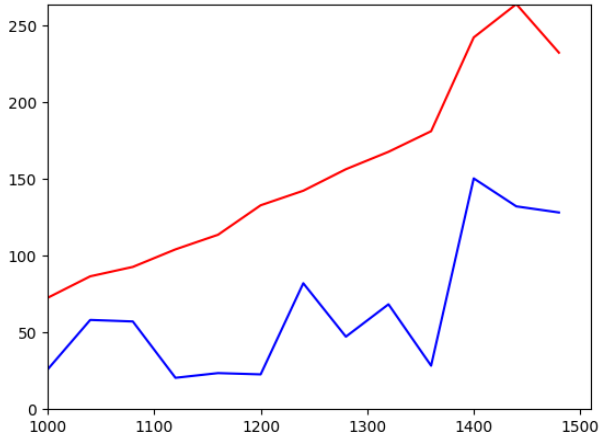


(Figure 16)



(Figure 16)

(Figure 17) If the dimension continuously rises to 1000-15000, EA finds the path much faster than BFS.



(Figure 17)

5.3 path redundancy

Besides the efficiency, the quality of solutions also needs to be analyzed. BFS always returns the shortest path, while EA does not. In the real world, a fitting curve can be calculated based on the path discovered by EA to remove redundancy. Considering the enormous extra complexity of BFS finding an optimal path, EA with curve regression is considerable.

6. Conclusion

When the dimension of the map is low, EA may take a longer time than BFS, conversely, EA is more effective. Meanwhile, EA performs well when search space is wide, while the complexity of BFS boosts too fast in this scenario. In conclusion, EA fits different environments in path planning problems with mild computation costs.

Reference

[1] Maze solver using BFS https://levelup.gitconnected.com/solve-a-maze-with-python-e9f0580979a1