Abdalrahman Mohammed

Genetic Algorithm for Maze Solving

Introduction

In this project, a genetic algorithm was implemented to solve a maze problem. The algorithm aimed to evolve a population of candidate solutions that could guide an agent from a start position to a goal position in a grid like maze. The genetic algorithm utilized concepts like selection, crossover, and mutation to iteratively improve solutions over multiple generations.

Maze representation

The maze is represented as a 2D list where each cell can either be a path or a wall:

- Path: Denoted by 1, representing cells the agent can move through.
- Wall: Denoted by 1000, representing cells the agent must avoid. The start position is defined at (0, 0) and the end goal at (12, 35). The agent's task is to navigate from the start to the goal using a sequence of movements.

Genetic algorithm structure

The genetic algorithm was designed to optimize the sequence of movements needed to solve the maze. Here is a breakdown of the key components of the implementation:

Initialization

- Moves: The agent can move in four directions:
  - U: Move up
  - D: Move down
  - L: Move left
  - R: Move right
- Population: An initial population of 500 candidate paths was generated, each with a fixed path length of 80 moves. Each path is a randomly generated sequence of moves.

Fitness function

The fitness function evaluates how good a candidate path is by assigning it a fitness score based on:

1. Manhattan distance: The fitness function rewards paths that bring the agent closer to the goal. The Manhattan distance between the agent’s position and the goal is calculated, and the score increases as the agent gets closer.
2. Penalties:
   - Revisiting positions: If the agent revisits a cell, a penalty of -150 is applied.
   - Hitting walls or going out of bounds: If the agent hits a wall or moves out of bounds, a penalty of -100 is applied and the path evaluation stops.

3. Goal reached: If the agent reaches the goal, a large reward of 10000 is given, significantly boosting the fitness of the path.

This combination of penalties and rewards ensures that the agent is motivated to find the shortest, most direct path to the goal while avoiding obstacles.

Selection

For the selection process, tournament selection was used:

- A subset of paths k = 5 was randomly chosen from the population.
- The path with the highest fitness score from this subset was selected as a parent. This approach helps retain high quality paths while still introducing diversity through randomness.

Crossover

The crossover function combines the moves of two parent paths to create two child paths. For each pair of moves from the parents:

- With a 50% probability, the moves from the parents are swapped to produce two children. This ensures that the children inherit characteristics from both parents, potentially leading to better paths.

Mutation

Mutation is used to introduce random changes in a path to maintain genetic diversity. With a mutation rate of 0.6:

- A random number of moves between 1 and 3 in the path are changed to random new moves.
- Additionally, with a 10% chance, a sub sequence of the path is replaced by a new randomly generated sub sequence. This helps explore new regions of the search space, preventing the population from getting stuck in less optimal solutions.

Elitism

The algorithm employed elitism to ensure that the best performing paths from each generation are carried over to the next. The elitism rate was set to 0.2, meaning that the top 20% of paths were preserved unchanged in the next generation. This ensures that the genetic algorithm retains the best solutions while still exploring new ones.

Main evolutionary loop

The main loop of the algorithm iterated over 1000 generations or until a solution was found. In each generation:

1. Evaluate fitness: The fitness of every path in the population is calculated using the fitness function.
2. Check for new best solution: The best path is updated if a better fitness score is found.
3. Elitism: The top 20% of the population is carried over to the next generation.

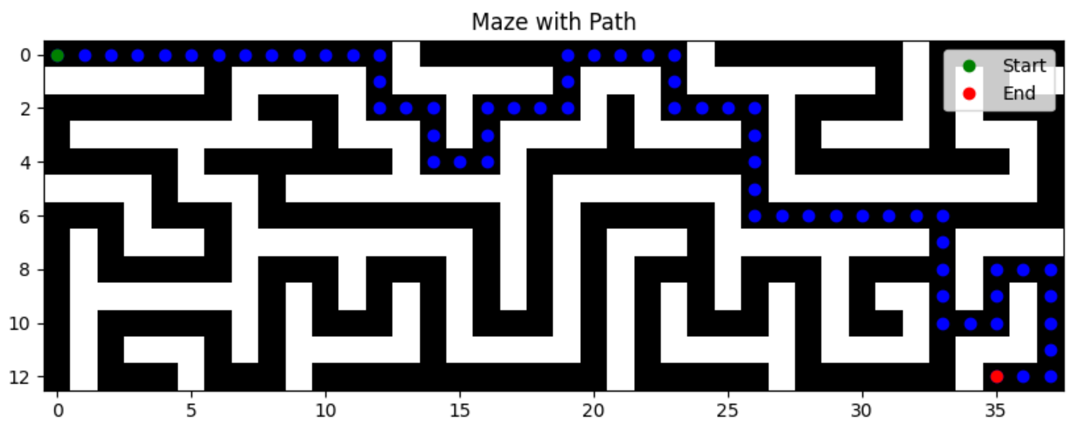
4. Crossover and mutation: The remaining population is generated through crossover and mutation of selected parents.
5. Update population: The new population replaces the old one, and the process repeats for the next generation.

Visualization

To visualize the results, a function was implemented to plot the maze and the path taken by the best solution:

- The maze is shown as a grid where walls are marked, and the path taken by the agent is plotted in blue.
- The start and end points are highlighted in green and red, respectively.

Results:



The algorithm successfully found a solution that reached the goal in Generation 333 with a fitness score of 10321,42. This path avoided obstacles, minimized revisiting cells, and efficiently reached the goal. The evolution process showed steady improvement in fitness over time due to the balance between elitism and diversity introduced by crossover and mutation.

Conclusion

The implementation of the genetic algorithm to solve the maze problem was successful. By using key concepts like selection, crossover, mutation, and elitism, the algorithm was able to explore the search space efficiently and find an optimal path to the goal. The balance between exploitation (elitism) and exploration (mutation) played a critical role in the success of the algorithm.