Seminar Assignment 1

Intelligent Systems - FRI

Gasper Spagnolo

November 11, 2022

1 Introduction

In the first seminar assignment, your goal is to use genetic algorithms to find a path out of a maze, represented as a vector of strings, where # characters represent walls, . represent empty spaces, and S and E represent the starting and ending points, as in a given example below:

```
maze = c("####E######",
         "##...#.####",
         "#..#.#.####",
         "#.##...####",
         "#.##.#..S##",
         "###########")
```

You can move through the maze in four directions, left, right, up, and down. In the example above, the shortest path from the starting position S to the exit E is composed of the following moves: left, left, up, left, left, up, up, up. In your solution, this should be represented as a string "LLULLUUU". Your task is to create a function that will be able to find path as short as possible out of any maze represented in such a way

2 Solution

2.1 Task 1

I decided to write this assignment in python using the pygad library becouse I am more familiar with this programming language.

2.1.1 Task description

Create a function that reads the 2D representation of a maze and returns the shortest path found by a genetic algorithm. To do this, you will need to:

- Read the map into a suitable format (for example, a matrix).

- choose a suitable representation of your solutions (the path). Hint: you don't need to use strings when working with the genetic algorithm. You can use numeric or binary representations for the GA function and then convert the result to a string as the final result.
- Define the fitness function. Make sure to penalise paths through walls - those are invalid solutions
- Run the genetic algorithm with suitable settings.

2.1.2 Read the map into a suitable format

I decided to read all the maps provided in the assignment into a list of lists. Each list represents a maze and each element of the list represents a row of the maze.

2.1.3 Choose a suitable representation of your solutions

I decided to use a binary representation of the solution same size as an original maze. Each bit represents a move. 0 means that the agent did not visit the cell and 1 means that the agent visited the cell. So if maze is of size N x M, the solution will be of size N x M. But there is no such thing as N-dimensional array that GA accepts. So I reshaped the matrix into a vector of size N * M and worked with that kind of solution.

2.1.4 Define the fitness function

This part was the most difficult for me. I maybe overcomplicated that part but at least it yields good results. Before runing the algorithm I have decided to construct a punish matrix, which is a matrix of the same size as the maze. Each cell in the punish matrix is evaluated before the algorithm starts. The evaluation is based on the position of walls and valid paths. So if there is a wall in the cell, the fitness value in that cell is set to some low scalar. If there is a valid move then the fitness value in that cell is high. So everytime the fitness function is called, the matrix product will be executed and some initiall fitness value will be computed asfollows:

```
fitness = np.sum(path * maze.punish_matrix.reshape(-1))
```

But though experimentation I found that this approach was not good enough so I modified the function by adding punsihment if the agent did not start at the starting position and if the agent did not end at the ending position. Still the results were not good enough so I decided to check if there is a valid path from the starting position to the ending position. If there is no valid path then I would punish the agent otherwise I would give him some reward. This approach yielded better results. But still I was not satisfied with the results so I decided to add some more punishes and rewards:

- Add a reward if agent finds a shorter path than the best path found so far.
- Update weights in punish matrix so that the agent will prefer to move on best path found so far.
- If the agent does not find any valid path until 80% of the GA iterations then activate critical search phase. That means that the agent will be rewarded if he finds **any** path from start to end, even if it maybe isn't the correct one. This way the weights are updated so that it converges to the correct path.

2.1.5 Run the genetic algorithm with suitable settings

I used the following settings wen running the algorithm:

- `number_of_genes = N * M`
  (if the maze is of size N x M) So the solution is a vector of size N * M.
- `num_of_generations = 1000`
  How many generations will the algorithm run.
- `sol_per_pop = 2`
  Number of solutions in the population.
- `num_parents_mating = 2`
  Number of solutions to be selected as parents in the mating pool.
- `keep_parents = -1`
  If -1, this means all parents in the current population will be used in the next population
- `allow_duplicate_genes = True`
  If True, then a solution/chromosome may have duplicate gene values.
- `mutation_type = "random"`
  Mutation type is random.
- `crossover_type = "two_point"`
  Applies the 2 points crossover. It selects the 2 points randomly at which crossover takes place between the pairs of parents
- `parent_selection = "tournament"`
  Selects the parents using the tournament selection technique. Later, these parents will mate to produce the offspring.
- `gene_type = int`
  We will be predicting integer values.
- `gene_space = [0,1]`
  Define binary subset to be gene space.
- `fitness_func = fitness_func`
  Specify fitness function.
- `parallel_processing = 4`
  Spawn 4 additional threads to speed up computing.

2.1.6 Results

1. On first maze I got a perfect score: *The shortest path is [(3, 1), (2, 1), (2, 2), (1, 2), (0, 2)]*

2. Same for the second one: *The shortest path is [(4, 5), (4, 4), (4, 3), (4, 2), (3, 2), (2, 2), (2, 3), (2, 4), (2, 5), (1, 5), (0, 5)]*

3. The third one had many problems and it did not want to converge to propper soluition.

4. The fourth one also found the solution pretty quickly. *The shortest path is [(5, 5), (4, 5), (3, 5), (3, 6), (3, 7), (3, 8), (2, 8), (1, 8), (1, 7), (1, 6), (1, 5), (0, 5)]*

Other mazes found also found some solutions, but they were not optimal. I think that the problem is that the mutation and crossover operators are not good enough. So I will try to improve them in the following sections.