Seminar Assignment 1

Intelligent Systems

October 19, 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.

You can work on the seminar assignment either individually or in pairs. The deadline for the assignment is 20. 11. 2022

2 Task 1 - Representation = 25%

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.

3 Task 2 - Crossover and mutation = 30%

The default mutation and crossover functions in R are not well-suited for this task because they do not necessarily return valid paths (for example, the mutation might introduce a move that goes through a wall). To fix this, modify the mutation and selection functions so that they take the walls into account. Additionally, try to create a starting population in a way that takes walls into account.

You can base your crossover and mutation functions on existing GA library functions, e.g., https://github.com/luca-scr/GA/blob/master/R/genope.R. Modify at least one crossover or mutation function in a way that makes them more suitable for this task.

For example, GA implements the uniform random mutation using the following function:

```
gareal_raMutation_R <- function(object, parent)
{
  #Select the parent vector from the population
  mutate <- parent <- as.vector(object@population[parent,])
  n <- length(parent)
  #Sample a random vector element that should be changed
  j <- sample(1:n, size = 1)
  #Change that element to a random number between the lower and upper bounds
  mutate[j] <- runif(1, object@lower[j], object@upper[j])
  return(mutate)
}
```

The function takes two paramters:

1. **object**, an object of class GA, the same class returned as the result of the `ga` function,
2. **parent**, a number of the object in the population that selected for the mutation.

The function returns the vector mutate, which represents the modified element.

Crossover functions are implemented in a similar way, with two major differences:

1. The parameter `parents` contains two (or more) parent numbers,
2. The function returns a list object in the following format: `list(children = children, fitness = fitness)`, where children are the newly generated elements and fitness are their fitness values, which can either be computed in the mutation function or set as NAs (`fitness = rep(NA, 2)`).

As long as your functions follow the described format, you can use them in the `ga` function by providing them as the mutation/crossover parameters.

4 Task 3 - Treasure = 20%

In Task 3, mazes also contain treasure (marked with T). For example:

```
maze2  = c("####E######",
           "##...#.####",
           "#..#.#.####",
           "#.##...####",
           "#T##T#..S##",
           "###########")
```

Your task is to modify your approach so that the solution returns as short a path as possible that also collects all the treasure.

5 Task 4 - Report and presentation = 25%

Present a report that describes your approach, shows highlights of your code, and presents the results. The results have to include performance comparisons between different settings of the genetic algorithm (different mutation, crossover and selection functions, different starting populations and so on). Make sure to evaluate your approach on different mazes, the one in the instructions is just an example. The mazes.r file on ucilnica contains several additional examples of various sizes and complexities. Find the largest size of a maze that can still be solved with your approach - feel free to create your own mazes if the example mazes are too small. Produce a graph to show how the maze size affects the running time of the genetic algorithm