Assignment 2

March 6, 2023

0. INTRODUCTION

Our next challenge is to enable our robot to navigate in the supermarket to find multiple products. The supermarket will be modeled as a labyrinth (further referred to as "maze"). In order to save time and energy, your bot should determine the fastest route, given a set of locations to visit within the supermarket. This problem is usually referred to as the *Traveling Salesmen Problem* (TSP) and can be solved with a *Genetic Algorithm*.

Before it can determine the fastest way to visit all locations, it should have an idea of what the distance is between these locations. To achieve this, you will implement *Swarm Intelligence*. Specifically, path finding based on *Ant Colony Optimization* (ACO).

Hence this assignment is split into 2 sub-problems: TSP and path finding. You will hand-in code individually for each part, but will have to hand in a single report answering all the questions. You can work through this assignment in the order detailed below.

We provide you with a template for the solution in Python. Information about the template is included in Appendix B

MAZE VISUALIZER

For this assignment, you can use the maze visualizer to create and test mazes. You can obtain this visualizer from Brightspace. The application can be used to visualize the routes computed. This way, you can check whether your answer is reasonable.

The application consists of two different modes: "editor" and "visualizer". The first is used to edit or create mazes; the latter can be used to visualize your routes in a given maze.

DELIVERABLES

Create a report to answer the questions from this document. Besides this, you have to clean up your code and **upload its final working version to the course Vocareum** in the corresponding assignment, **including a Jupyter Notebook** which (1) uses ACO to find the best paths in the hard maze (2) executes the Genetic Algorithm using the data from a persistence file. You should check if the notebook works as expected before pressing the `Submit` button. Furthermore, we ask you to **deliver the following files via Vocareum**:

- a report in PDF format with answers to all questions. Your report must be no longer than 9 pages (including visuals) + the title page and references. Please structure your report based on the numbered questions in this document:

  `"<group_number>_report.pdf"`

- a route file for the `Easy`, `Medium`, and `Hard` mazes (the `Insane` maze is not required):

  `"<group>_easy.txt"`

  `"<group>_medium.txt"`

  `"<group>_hard.txt"`

- a final action file for Part 1 (see route syntax in Section 1):

  `"<group_number>_actions_TSP.txt"`

Note: Your code will not be directly graded - do not rely on it to support your answers. Note that we will check your code for irregularities so remember to keep it code clean, use proper indentation, provide useful comments, name your variables logically, etc.

To speed up the grading process, we want you to deliver your work exactly as outlined above. **If (a) your files are not of this format or (b) the main Jupyter notebook does not run, your work will not be graded.**

Fraud will not be tolerated. You are allowed to discuss concepts and ideas with colleagues from other groups, but you are not allowed to share code outside your own group. The same applies to submissions from previous years. You are highly encouraged to make use of the provided GitLab repositories.

You may ask questions on Answers-EWI (while keeping in mind that it's a public forum, so please don't share partial solutions), or to a TA during the labs.

The deadline for the assignment is Monday **20th of March 2023 at 18:00**.

1. PART 1: THE TRAVELING ROBOT PROBLEM

In the first part of the assignment you will be tackling a modified version of a classic AI problem, commonly known as The Traveling Salesman Problem (TSP) [1].

As you can imagine (and probably know from your own experience), there are a lot of possible routes to take through a supermarket if you want to pick up several products. For 3 products, there would be $3 \times 2 \times 1 = 6$ possible routes. This number increases exponentially. For 20 products, we already have $20! = 2.4 \times 10^8$ possible routes! Clearly, brute-force trying out all these routes to find the fastest would take too much computational time. Instead, you will be implementing an intelligent way of converging towards a (nearly) optimal path.

WHAT WILL YOU NEED

On Brightspace and Vocareum you will find the data needed for this part of the assignment:

- the `Hard` maze (you can read about its representation in Appendix A);
- the file containing optimal paths between all pairs of products (your input);
- the x and y coordinates of a list of products, which your robot must go and pick up;
- the route visualizer (*optionally*)

ROUTE SYNTAX FOR PART 1

You need to encode your results (routes) by their length, starting location, and a sequence of "actions" to be taken by the robot. The header of the file, the total route length, is stored as an integer. This value is equal to the number of actions. A "`;`" should follow this value. The starting point should be given by two number: the x-coordinate and the y-coordinate, separated by a comma ("`,`"). A semicolon ("`;`") should follow the y-coordinate.

There are five possible actions: steps `East`, `North`, `West` or `South`, and `take product`. The first four actions are encoded using an integer;

- 0 = East
- 1 = North
- 2 = West
- 3 = South

Within the steps encoding, all characters other than $\{0,1,2,3\}$ are invalid. Again, each action should be followed by a semicolon ("`;`"). You can test the validity of the route file with the visualizer. The provided templates will put the files in the right format for you.

[1] `http://en.wikipedia.org/wiki/Travelling_salesman_problem`

The last action indicates when a product has to be picked up by the robot. This is done by inserting the line `take product #<Product Number>;` in between the steps actions. Part of a result file might thus look like:

```
83;
0, 1;
3;
3;
0;
take product #6;
3;
2;
2;
take product #15;
3;
1;
1;
1;
take product #4;
1;
0;
(...)
```

The provided templates contain export functions to get the appropriate file format needed.

PROBLEM ANALYSIS

To get a good understanding of the nature of our problem, start by analyzing its characteristics. Your robot will have to pick up a number of different products from all over the supermarket, which can be found at certain locations.

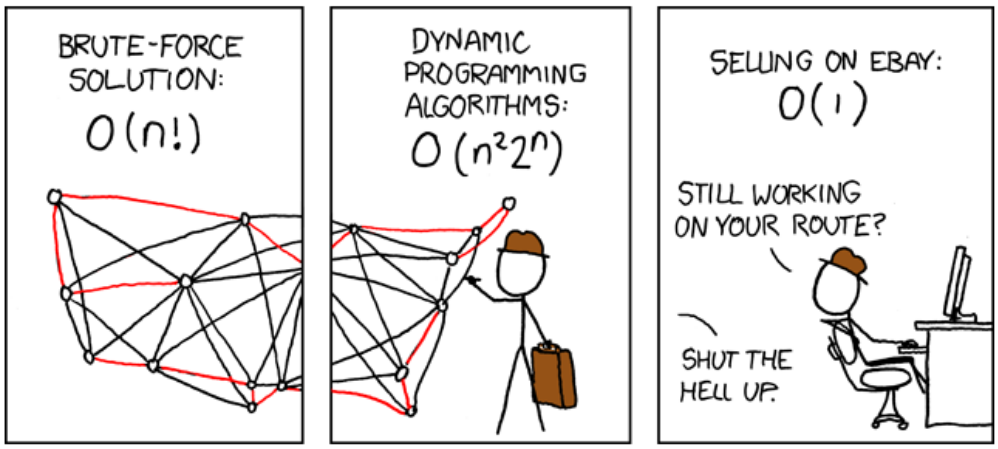
Think about the following questions and answer them in your report.

1. (1 point) Look into the literature and explain how the TSP problem is usually defined.
2. (1 point) In a classic TSP, all cities (nodes) are singularly connected to all other nodes and relative distances are known and symmetric (a weighted complete graph). Give at least two ways in which our problem is different.
3. (2 points) Why are computational intelligence techniques appropriate to solve the TSP? What are the characteristics which let us to tackle typically intractable problems?

A GENETIC ALGORITHM

A classical method of solving a TSP are genetic algorithms. This technique is part of the evolutionary computation methods. Biological evolution is sometimes described as non-random selection through random gene-mutations. Genetic algorithms can be described using the same nomenclature: random search through non-random selection of the solution space. Since this space is too big to verify all possible outcomes, we need an algorithm which directs us towards viable options. Answer the questions **based on your implementation**.

4. (1 point) What do the genes represent? How will you encode your chromosomes?
5. (1 point) Which fitness function will you use? Why is this a suitable choice?
6. (1 point) How are parents selected from the population?
7. (3 points) What are the functions of your genetic operations (cross-over, mutation, ...)? How do you prevent points from being visited twice?
8. (1 point) How do you prevent local minima?
9. (1 point) What is elitism? Have you applied it? Why (not)?



source: http://xkcd.com/399/

2. Part 2: Path Finding Through Ant Colony Optimization

In the second part of this assignment you will teach your robot how to move through a maze and implement a method for finding a quick route between a specified start and end point. The goal of the assignment is to use the methods learned in this course to tackle the problem, hence you are **not** allowed to use search algorithms like Dijkstra or A*.

2.1. Route Syntax for Part 2

The route syntax is the same as for Part 1; however, you will not be making use of the `take product` actions. In other words, the only available actions are `East`, `North`, `West` or `South`.

2.2. Ant Preparation: Observe The Problem

Before you go ahead and implement the ant algorithm you learned in class, you should stop and think for a second: "what kind of features/problems can the maze contain and which should we take into account when implementing the algorithm?".

10. (1 point) What is the purpose of Ant Colony Optimization? When is it applicable/used?
11. (1 point) Make a list of the features you can expect a maze might have. Features that increase the difficulty of "finding the finish line" and require creative solutions, for example loops. Name **at least 2 other** features.
12. (2 points) Give an equation for the amount of pheromone dropped by the ants. Explain why ants need to drop the pheromones in the maze.
13. (2 points) Give an equation for the evaporation; it should contain variables which you can use to optimize your algorithm. How much pheromone will evaporate in every iteration? What is the purpose of pheromone evaporation?

2.3. Implementing Swarm Intelligence

Now, implement the **standard** ant algorithm. The output of your algorithm should be a route between the given start and end point. You may want to already take the route syntax into account. Your algorithm should satisfy a few conditions:

I. At least the following parameters should be **easily** modifiable (e.g. in the file header):
   a) maximum number of iterations (to prevent infinite loops);
   b) number of ants patrolling in each iteration;
   c) amount of pheromone dropped by ants;
   d) evaporation parameters;
   e) convergence criterion.

II. The beginning- and ending point of the route should be a variable. Your code should be able to find a short route between any two points (if such route exists).

At this point it is sufficient to assume that the maze does not have any of the hard features you thought of in the first question. Simply implement the standard ant algorithm. You will update your algorithm later to deal with the harder features, but its useful to already test how the ants behave. Still, its good programming practice to have the future upgrades in the back of your mind while writing your basic algorithm to facilitate later adjustments.

14. (4 points) Provide a short pseudo-code of your ant-algorithm at this stage. If you added any extra functionality to the normal algorithm, please mention and explain it briefly.

2.4. Upgrading Your Ants with Intelligence

You will now adapt the ant algorithm to deal with features like open areas and any other tricky features you came up with. There are many possible approaches here – it is a good idea to create small test mazes to see how your code behaves when these features occur.

Note: We expect you to keep with the biologically-inspired nature of the algorithm. When designing your improvements, think about the actual behaviour of animals such as ants. Some adaptations will be considered "cheating" within the scope of this assignment. For instance, when your ant enters the maze it should not know where the exit is located, so any form of directed search will not be accepted. However, it is accepted if your ants use some form of memory. If you are in doubt about your improvements, consult the TAs in the lab.

15. (4 points) Improve the ant algorithm using your own insight. Explain which problems you are tackling and how? We would like to see at least three **meaningful** improvements (as an example you could consider giving your ants some form of memory). Limit yourself to 1 page A4 (including aiding figures).

2.5. Parameter Optimization

We have provided 3 different grading mazes for you to work with: `Easy`, `Medium` and `Hard`. You are expected to run your code on each of these mazes to see how your code behaves for an increasing maze complexity. While doing so, you should tweak the algorithm parameters to make your code converge quickly.

The optimal set of parameters (evaporation constant, amount of pheromone dropped, etc.) depends strongly on the precise shape of the maze. A bigger maze will typically have different optimal parameters, but so will a maze with other features (e.g., many open areas).

16. (3 points) Your task is to find a decent set of parameters for each of the grading mazes. You may do so by varying the parameters and subsequently running your algorithm. If your algorithm converges fast to a good route, your parameters are decent. What does "converging fast" mean? Figure that out by varying the parameters and explain it. Report your approach to tuning the parameters. Assist your text with graphs showing

the relationships between the parameters and the speed of convergence. Limit yourself to 1 page A4 (including aiding graphs but we like nice informative graphs, so use them!).

17. (2 points) Using your answer to the previous question, can you say something about the dependency of the parameters on the maze size / complexity? Aim at $\approx \frac{1}{4}$A4.

2.6. The Final Route

Run your code using your decent set of parameters on each of the grading mazes. Output your route as described in the Route Syntax section (you can verify the correctness by opening your route in the visualizer). See also the "Deliverables" section.

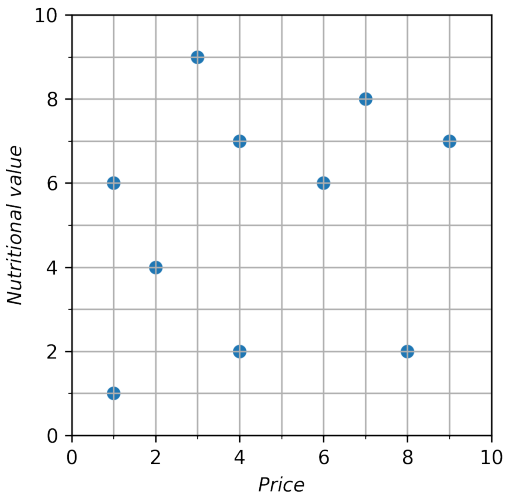
2.7. Synthesis

We will now combine the two parts of the assignment. Use your Ant Colony Optimization algorithm in the `TSPData.py` file to generate your own estimate of distances between the start and each product, each pair of products, and each product and the end. Use this file to re-run your genetic algorithm and find the optimal order of actions.

18. (2 points) What are the differences (length, order of items, etc.) between the current path and the path found by your algorithm in Part 1? Which solution is better?

2.8. Pen and paper

Finally, we have two questions for you to be solved on paper which serve as the preparation for the final exam. They are independent from the rest of the questions in this document. You are asked to submit only one set of answers but we strongly suggest that each team member solves them separately, so that you all can practice with the calculations.



19. (3 points) Imagine that our robot attempts to select the best set of foods based on their price and nutritional value (see the plot above). Which of the variables do you think we should maximize or minimize? What are the Pareto frontiers? If the robot can carry only four products, which of them will it choose according to the NDSGA-II algorithm? Justify your answers.

20. (3 points) Assume that the daily caloric intake of a person is modeled as:

$$C(w,h) = 20w + 70h - wh + 500$$

with h representing height (in metres) and w representing weight (in kilograms). We are interested to find the *maximum* daily caloric intake in a specified range. To that end, we can apply Particle Swarm Optimization. Calculate the first iteration of the PSO algorithm given the following information:

- $w \in [50,\ 100]$
- $h \in [1.2,\ 2]$
- number of particles: 3
- constants for all particles: $c_1 = c_2 = 0.5$
- random scale factors for all particles: $r_1 = r_2 = 1$
- initial velocity for all particles: $v_w = v_h = 0$
- initial positions:
  - $w_1(0) = 85$, $h_1(0) = 1.9$
  - $w_2(0) = 60$, $h_2(0) = 1.6$
  - $w_3(0) = 75$, $h_3(0) = 1.7$

what are the local bests of the particles after the first iteration? Show your calculations.

A. GRADING MAZES

Besides creating, analyzing and testing mazes yourself, you will need to run your code on each of our grading mazes. These mazes are available on Brightspace and in Vocareum. There are 5 mazes available: `Toy`, `Easy`, `Medium`, `Hard` & `Insane`. Note that it's not mandatory to solve the `Insane` maze, but you can use it to show off the efficiency of your code.

The mazes are ASCII encoded matrices of 1s (accessible) and 0s (inaccessible). Your robot is allowed to walk single discrete steps within this matrix; however, only the tiles that are accessible. For your convenience, the maze files start with two numbers which indicate the width and height of the matrix: a 10 by 20 matrix – 10 rows by 20 columns – is indicated as "20 10". You can open the maze files in any text editor to see what they look like.

```
20 10
1 0 1 1 1 1 1 1 1 1 0 0 0 1 0 1 0 1 1 1
1 0 1 1 1 1 0 1 0 1 1 1 0 1 0 1 0 1 0 1
1 0 1 1 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
1 1 1 0 0 1 0 1 0 1 0 1 0 1 1 1 1 1 0 1
0 0 0 0 0 1 1 1 0 1 0 1 1 1 0 1 0 1 1 1
1 1 1 1 1 1 0 1 0 1 0 0 1 0 0 1 0 0 0 1
1 0 1 0 1 0 0 1 0 1 0 0 1 0 0 1 0 1 1 1
1 1 1 0 1 0 0 1 1 1 0 1 1 0 1 1 1 1 0 1
0 1 0 0 1 1 1 1 0 1 1 1 0 0 1 0 0 1 0 1
0 1 1 1 1 0 0 1 1 1 0 1 1 1 1 0 0 1 1 1
```

Accompanied with each grading maze is a coordinate file. This file consists of 4 numbers (2 sets of coordinates). The first two integers correspond to the starting position in the maze: "x, y;". Note that these are x and y coordinates, not a row & column number! The second two integers correspond to the ending point of the maze. Hence for a 50x50 maze, starting at the top left and finishing in the bottom right, the coordinate file would read:

```
0, 0;
49,49;
```

B. INSTRUCTIONS FOR TEMPLATE

We provide you with a Python template to get started. There are a number of stubs/unimplemented functions that you will need to implement to solve the assignment. You can create extra functions/modify existing functions as you see fit. You can import the Python project in an IDE like PyCharm or Visual Studio Code. The classes `AntColonyOptimization`, `GeneticAlgorithm` and `TSPData` have a main function that can be run.

We also provide you with an overview of the classes available in the template.

B.1. PART 1

GeneticAlgorithm driver function for the Genetic Algorithm. Takes a file that contains a 2D-matrix containing the routes between products and attempts to optimise those. You will need to implement the genetic algorithm yourself.

TSPData builds and stores the data for all the shortest paths between the products. You will use it only after completing Part 2 of the assignment based on your implementation of the AntColonyOptimzation. You will only have to tune the parameters. It can also write the final action file given a solution to the TSP problem.

B.2. PART 2

Ant represents an ant finding a path through the maze. You will need to implement logic how an ant finds a route through the maze.

AntColonyOptimization driver function that finds an optimal route between two points. Allows you to tune the parameters of the algorithm. You will need to implement the creation of a maze, the ants to allow them to run through the maze.

Coordinate represents a coordinate. All functions are implemented.

Direction represents the directions an Ant can take. All functions are implemented.

Maze represents a maze. You will need to implement the initialization, addition and reset of the pheromones. Besides this you will need to implement a function that returns a function that returns SurroundingPheromone for a given coordinate.

PathSpecification A class representing the pair of coordinates that indicate the start and end points of a route. Used as initialization for shortest paths/ant colony optimzation.

Route represents a route. All functions are implemented.

SurroundingPheromone represents the surrounding pheromone for a given coordinate in the maze. All functions are implemented.