Assignment 2 - Report

CSE2530 Computational Intelligence

1.1.2. ANT PREPARATION: OBSERVE THE PROBLEM

1. Make a list of the features you can expect a maze might have. Features that increase the difficulty of ’finding the finish’. Features that will require creative solutions. For example; loops. Name at least 2 other features.

a. If there are loops in the route.
b. If many routes use a link which isn’t part of the shortest route.
c. If there are two routes with an equal probability of being picked.
d. If there’s a dead end.

2. Give an equation for the amount of pheromone dropped by the ants. Answer the questions “Why do we drop pheromone?” and "What is the purpose of the algorithm?".
We use this formula from the lecture slides which computes the amount of pheromone on path ij:

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}$$

Why do we drop pheromone?
It enables the ants to explore the environment effectively. The ants release pheromone on their path, proportional to the length of the path. Shorter paths will have more pheromone on them, which the next ants will prefer to take since the probability that an ant takes a specific route depends on the pheromone on it.

What is the purpose of the algorithm?
Find the shortest path to a given goal.

3. Give an equation for the evaporation. How much pheromone will vaporise every iteration? This equation should contain variables which you can use to optimize your algorithm.
Evaporation is accounted for in the formula given in question 2, and it’s given by the evaporation constant ρ. This is a variable which we can tweak accordingly.

What is the purpose of pheromone evaporation?
If there is a long and popular path, the evaporation will cause them to have a hard time maintaining their pheromone trails. (Since we're looking for the shortest route, a long path is undesirable).

1.1.3. IMPLEMENTING SWARM INTELLIGENCE

4. Give 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 them briefly.

```
The ant starts in the start position
For every iteration
    If an ant is at the goal: return the taken route
    Call Get_surrounding _pheromone to get the pheromones in the surrounding directions
    for(every direction possible to take from the current location)
    Calculate the probability for taking that path:
    (pheromone^alpha * (1/length of path)^beta )/(sum of total pheromone ^alpha * (1/length of the path)^beta)
    Pick the direction stochastically (Drawn from the distribution)
    Update the current location according to the chosen direction
Return the taken route
```

1.1.4. UPGRADING YOUR ANTS WITH INTELLIGENCE

5. Improve the ant algorithm using your own insight. Describe how you improved the standard algorithm; which problems are you tackling and how? Limit yourself to 1 A4 text
We'll be tackling the problems mentioned in question 1.

The first problem we mentioned is the loops. Loops cause problems since taking a loop won't contribute towards a shorter route, it will just extend the route.
A quick note to get an intuition for our solution: you can't have a loop without visiting the same location twice. This means that if you were in location x, continue your path and end up in location x again, you can remove the path taken between the two consecutive x's and this will not extend your route, it could only shorten it.

Practically, the way we tackle this is by keeping an array *visited* with the visited route and updating it with every step we take. In every iteration, we check if the current location has been visited before.
If it has been visited before:

1. Store the current location x (which is the second time that location has been visited since it has already been visited in the past)

2. Update the parameter *not_dir* to be the direction taken before arriving at the current location x.
3. Go a step back
4. Remove the last visited location from the *visited* array.
5. Repeat steps 2, 3 and 4 until we're back to the first instance of location x.

The second problem we tackle is If many routes use a link which isn't part of the shortest route. But turns out that this will be taken care of by evaporation. Since a longer path (which shouldn't be a part of the shortest route) will have a harder time keeping his pheromone because it will be evaporated.

The third problem we mentioned is that if two routes have an equal probability of being picked. This turned out not to be an issue since we pick the direction stochastically, i.e. drawn from the distributions. (This means that even if the path i has a lot of pheromone and path j has a little, path j might still be picked, just with a smaller probability.)

The final problem we mentioned is dead ends. The way we took care of this is by noticing the following: if the amount of possible directions the ant can take is 2 or lower, the ant might be in a dead-end or walking towards a dead end. The way we resolve this is by backtracking. If the total surrounding pheromone is 0, we have hit a dead end and have to go back, and we do this with the *backtrack* function. While the amount of possible directions is less than 2 (so 0 or 1): update the *not_dir* variable to be equal to last taken direction, update the *blocked* dictionary, move a step back and remove the last visited location from the *visited* array. This just means that if an ant has no available moves it should retrace its steps until it does.

1.1.5. PARAMETER OPTIMIZATION

6. 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 is 'converging fast'? Figure that out by varying the parameters. Report your tactic upon how to vary the parameters. Assist your text with graphs showing relationships between the parameters and the speed of convergence.

Since we thought that putting up all the graphs would take up way too much space, we stuck to these 3 graphs which hold true in general.

First we have the evaporation factor.
In figures 1 and 2 we varied the evaporation factor for the hard and medium maze respectively, while keeping the amount of ants per generation and amount of generations constant.

According to figure 1, the evaporation factor doesn't really affect the length of the route for the hard maze, with them basically being interchangeable. The same holds true for the easy maze.

In contrast, the evaporation factor does play a role when it comes to the medium maze, with an evaporation factor of 0.3 creating the shortest route compared to the other values.

Secondly we have the amount of ants per generation.
For this we have figure 3 which portrays the effects of varying the amount of ants per generation for the medium maze, when the evaporation factor and amount of generations are kept constant. According to this figure, 15 ants per generation creates the shortest route compared to the other values. We noticed how harder the maze became, how much more favorable it became to have more ants per generation. This is because the maze becomes bigger and more cumbersome to explore, the probability of fewer ants to find the optimal path is smaller than if there are more ants exploring at once. This doesn't hold quite as true for the easy maze, since fewer ants are needed to effectively explore the maze.

Finally, we have the amount of generations. We didn't add a graph for this, because to our surprise, the graphs showed that the amount of generations didn't change much when it comes to finding a shorter path. There was a small improvement when it comes to the hard maze, as expected, but nothing notable.

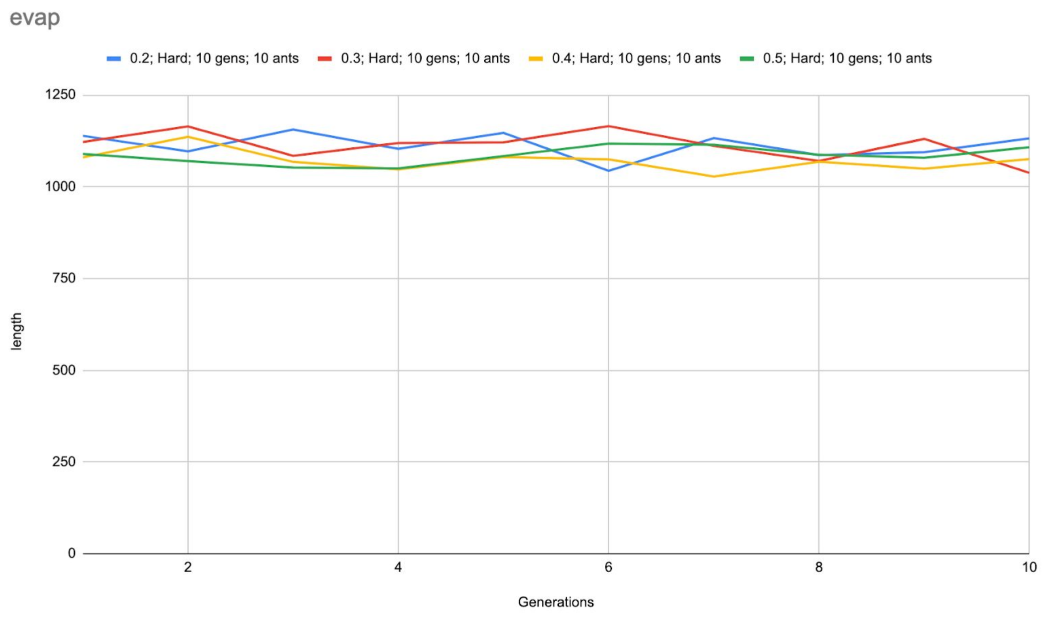

Figure 1

Varying evaporation factor - hard maze with 10 ants per generation, for 10 generations.

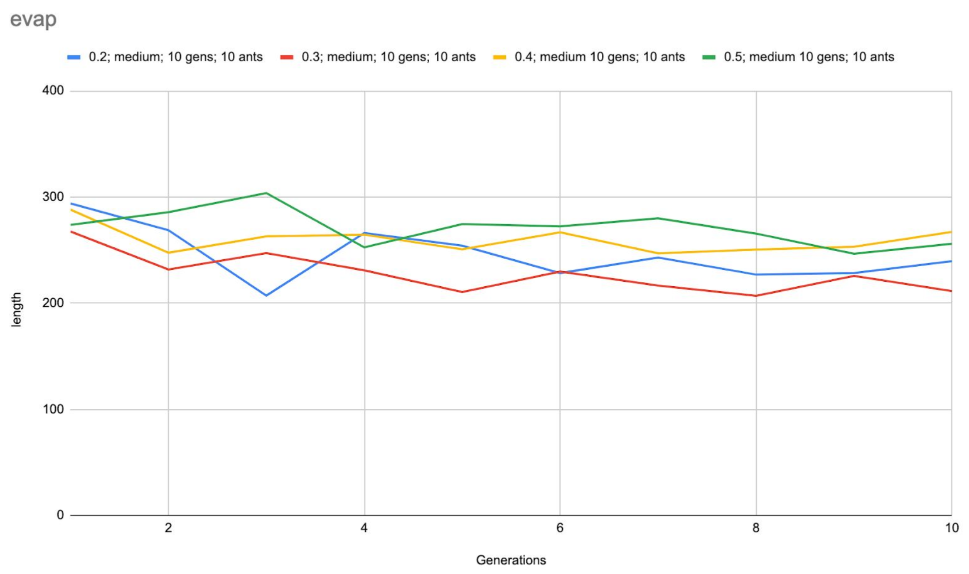

Figure 2

Varying evaporation factor - medium maze, 10 ants per generation, for 10 generations.

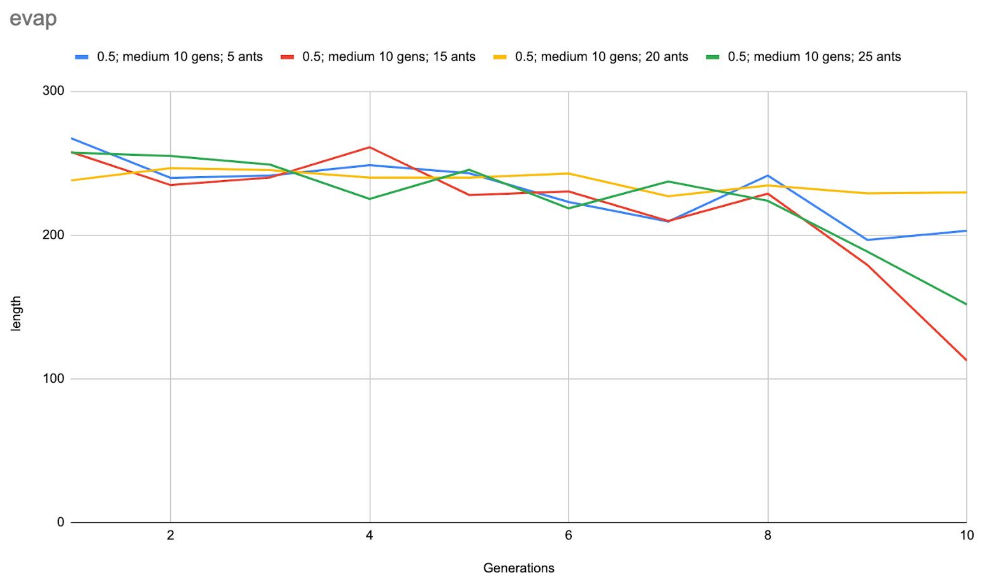

Figure 3

Varying amount of ants per generation, medium maze with evaporation factor equal to 0.5 and 10 generations of ants.

7. Using your answer to the previous question, can you say something about the dependency of the parameters on the maze size/complexity?

The number of ants per generation, the amount of generations and evaporation factor can all be tweaked to find an optimal solution for each of the mazes, as we have seen here above.

The evaporation factor had a small effect when it came to the easy and hard maze, but had a pretty considerable effect on the medium maze. We conclude that it’s still a good feature to have, since it has some very important implications for the algorithm, but the value of it doesn’t really matter.

But the amount of ants per generation and amount of generations became more prominent parameters when it comes to finding the shortest route in a harder maze. Those parameters didn’t matter that much when it came to the easy maze, but they really took effect when it came to the medium and hard maze. That’s because these mazes are bigger and the probability of fewer ants finding the optimal route is lower than when you let more ants find the optimal route at once, that way we have a higher probability of the ants finding an optimal route. Also, the amount of generations didn’t have as big an effect as we thought, but we still think that if the maze is bigger and more complex, more generations would be favourable, since it gives the ants more time to find the optimal route.

1.2. PART 2: THE TRAVELING ROBOT PROBLEM

PROBLEM ANALYSIS

8. Define the regular TSP problem.
In the original Travelling Salesman Problem, a person has to find the shortest route to visit a given list of cities, each exactly once. (and return to the original city he departed from) It’s a NP-hard problem.

9. 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). How is our problem different?
We don’t know the distances ahead of time so we have to calculate it first with the first part of this assignment, namely we have to calculate the shortest routes between all the product locations.

10. Why are Computational Intelligence techniques an appropriate tool for solving a TSP?
Because the ants will progressively, and successfully, find shorter and shorter routes until a near-optimal solution is found. Additionally, popular links which aren’t part of the optimal route will eventually be replaced by shorter ones because of evaporation → Best links survive. Finally, it’s flexible. Because the ants are continuously exploring the different paths, the pheromone paths provide backups which can be used in case a link breaks down.

11. What do the genes represent and how will you encode your chromosomes?
Chromosomes will be bit-string encoded in the candidate solution in which genes represent the individual bit in the chromosome.

12. Which fitness function will you use?
Our fitness function is a min function which we will be using. It chooses the minimum distance to each path and that's its fitness value.

13. How are parents selected from the population?
We pick two random numbers between 0 and the max cumulative fitness, followed by checking the slice it falls under and then selecting the corresponding pair chromosomes.

14. The functions of your genetic operations (e.g. crossover, mutation).
With probability p_c, we will pick a random crossover point and then apply a percentage addition, or subtraction, among the two chromosomes to update its values. After doing this for all pairs, we will apply mutation to flip the final bits in the chromosomes with probability p_m.

15. How do you prevent points from being visited twice?
It should be noted that the repetition of the product locations cannot be fully avoided in genetic algorithms. Moreover, repetition can indicate that the result is converging to a final solution. In order to reduce repetition, the crossover and mutation probabilities can be increased and this can lead to a more robust solution.

16. How do you prevent local minima?
In general, optimization solving algorithms converge to a local minimum. To get out of this local minimum, we made use of mutations. Mutations are applied to some individuals of a generation. Usually, mutations will be bad and make the result worse and they will not be selected for the next generation, but sometimes, a mutation causes an individual to get close to a different (and sometimes better) local minimum.

17. What is elitism? Have you applied it?
Elitism is the process of copying a small subset of the fittest candidates, unchanged, into the next generation. This can boost the performance by ensuring that the Ants do not waste time by re-discovering previously discarded partial solutions that were not efficient. Candidate solutions that are preserved unchanged through elitism remain eligible for selection as parents when breeding the remainder of the next generation. We haven't applied elitism due to time constraints but we will keep it in our mind for the future as it's a powerful concept.