Report Assignment 2

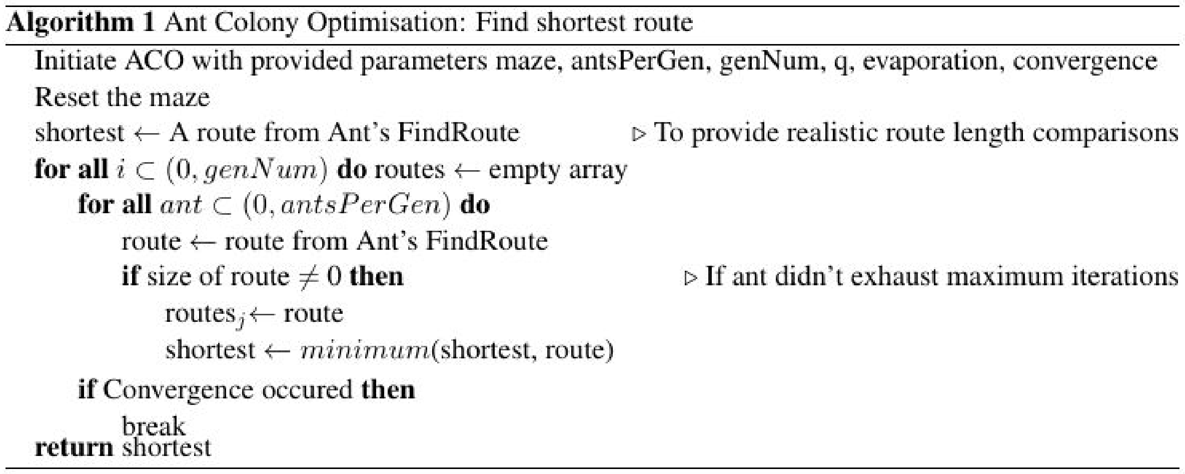
Group 28

1.1 Part 1

Assignment 1.1.2

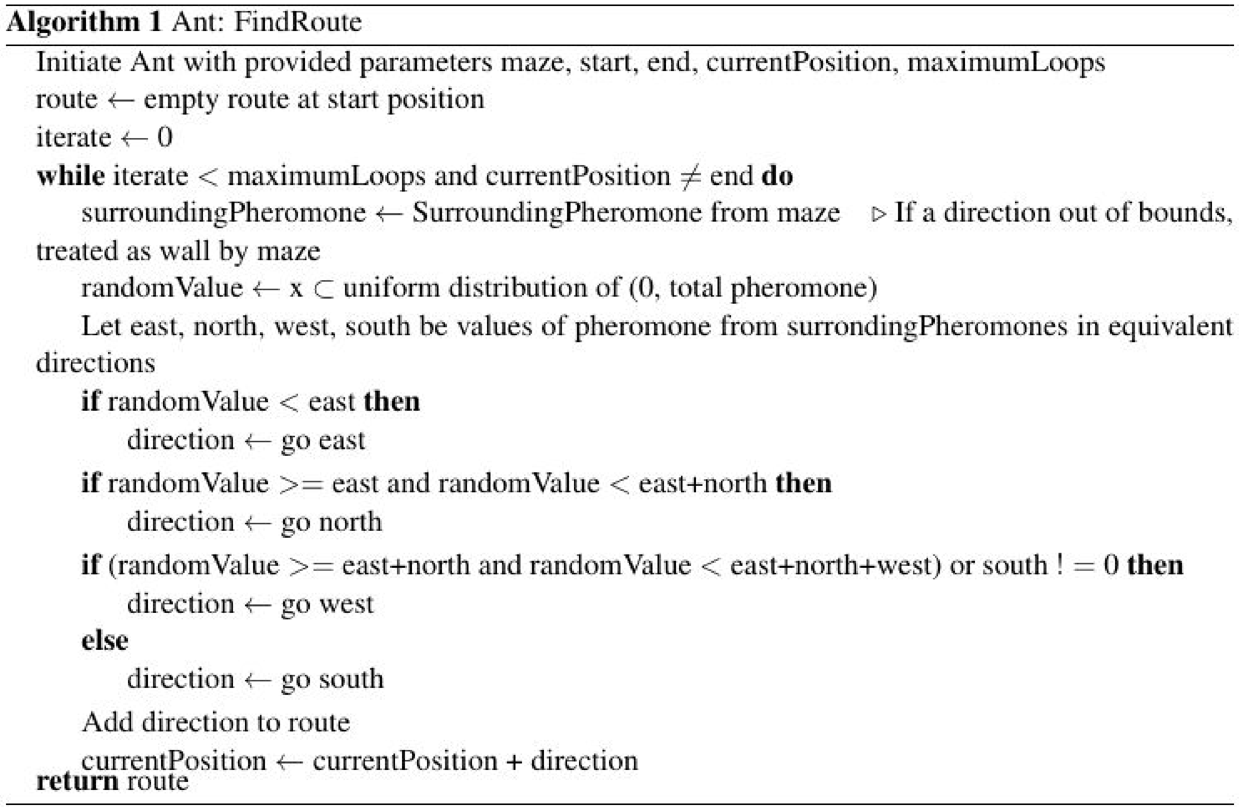
1. Loops, dead ends, and mazes such that goal is not accessible from the specified starting point.
2. $$\tau_i = \sum_k \Delta\tau_i^k$$
   Total pheromone on path i
   $$\Delta\tau^k = Q \times \frac{1}{L^k}$$
   amount of pheromone dropped by ant k on a route of length Q
   As ants walk around looking for their destination, they drop pheromone. The ants drop pheromone to remember the length and favourability of a given path.
   If a path has more pheromone, it has a higher probability of being followed by ants.
   Eventually the shortest path will be the path with the strongest pheromone trail.
3. $$\tau_{ij} = (1-\rho)\cdot\tau_{ij} + \sum_{k=1}^{m}\Delta\tau_{ij}^k$$
   Every iteration there will evaporate an amount of pheromone proportional to ρ, meaning that what is left on the path is what is left from evaporation plus the sum of all the pheromone released on that link by all the ants that passed it by this round.
   The purpose of pheromone evaporation is so that long links that are part of shortest path are more likely to be included and short links that aren't part of the shortest path are more likely to be excluded.
   ρ : evaporation constant
   Tij: pheromone on path from i to j
   m: number of ants
   k: current ant

Assignment 1.1.3



Algorithm 1 Ant Colony Optimisation: Find shortest route

```
Initiate ACO with provided parameters maze, antsPerGen, genNum, q, evaporation, convergence
Reset the maze
shortest ← A route from Ant's FindRoute                ▷ To provide realistic route length comparisons
for all $i \subset (0, genNum)$ do routes ← empty array
    for all $ant \subset (0, antsPerGen)$ do
        route ← route from Ant's FindRoute
        if size of route $\neq 0$ then                  ▷ If ant didn't exhaust maximum iterations
            $routes_j$ ← route
            shortest ← $minimum$(shortest, route)
    if Convergence occured then
        break
return shortest
```



Algorithm 1 Ant: FindRoute

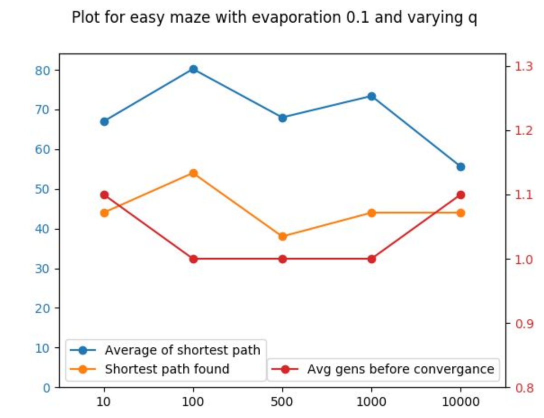
```
Initiate Ant with provided parameters maze, start, end, currentPosition, maximumLoops
route ← empty route at start position
iterate ← 0
while iterate $<$ maximumLoops and currentPosition $\neq$ end do
    surroundingPheromone ← SurroundingPheromone from maze    ▷ If a direction out of bounds, treated as wall by maze
    randomValue ← x $\subset$ uniform distribution of (0, total pheromone)
    Let east, north, west, south be values of pheromone from surrondingPheromones in equivalent directions
    if randomValue $<$ east then
        direction ← go east
    if randomValue $>=$ east and randomValue $<$ east+north then
        direction ← go north
    if (randomValue $>=$ east+north and randomValue $<$ east+north+west) or south $! = 0$ then
        direction ← go west
    else
        direction ← go south
    Add direction to route
    currentPosition ← currentPosition + direction
return route
```

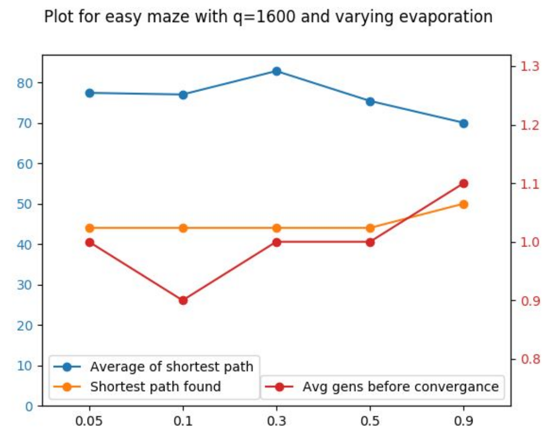
Assignment 1.1.4

5. The standard algorithm did not take into consideration when there were (1) loops, (2)dead ends and (3)unreachable goals in the maze. We also had the problem of an (4)ant walking back to the position it came from, because the pheromone of the previous position was higher. We improved our algorithm by taking these extra difficulties into consideration.
For the first two problems and the problem of the ant walking back where it came from, we decided on adding a list to keep track of all the coordinates of the positions the ant has walked. To deal with not allowing the ant to turn back, we made it such that the ant always assumed the pheromone in the direction it came from is 0, without actually changing the underlying pheromone array. This changed the ant route in r=the easy maze from a few hundreds to few tens of steps.
To deal with dead ends, we designed an algorithm so that if ant thinks all pheromone around it is 0, the ant will step back and set the pheromone in the pheromone array to 0 on the tile it was previously, until the ant reaches a part of the route where it can take a different road. The dead-end steps are erased from the position list and not considered part of the shortest route. Any next ant, in this or next generation will not go up the dead end, as it perceives it as a wall.
To deal with loops, we expanded the solutions for two previous problems, so that the ant is not allowed to take a step to a tile that was already visited, and it perceives all visited tiles as walls of the maze, with pheromone 0. If the ant cannot go anywhere without creating a loop, this is solved in the same way as dead end, allowing the ant to step back and explore a different route, while setting the tiles it backs down from as 0. It might make the shortest route impossible to achieve, but since ACO only finds short, not shortest paths, we decided that this sacrifice is acceptable.
We didn’t create a fix for unreachable goals. If an unreachable goal occured, the ant would explore all possible tiles and freeze. We assumed that a maze with path specification that has an unreachable goal doesn’t have a solution anyway, so it’s not fixable. The program would have to be halted manually.

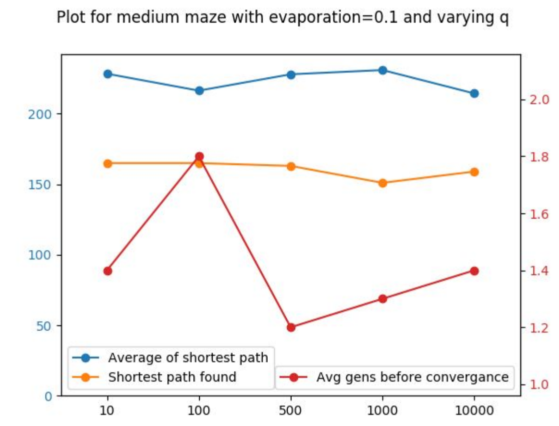
Assignment 1.1.5

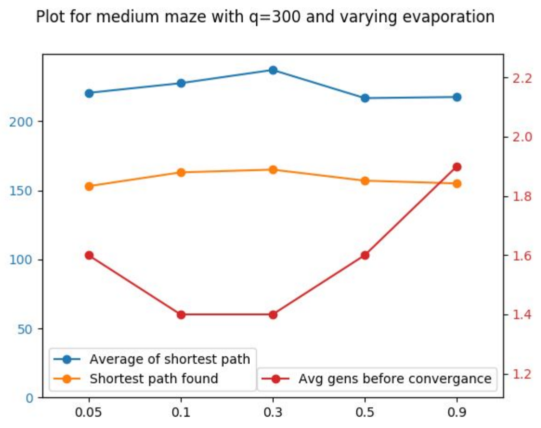
6. As a measure of convergence to use “generations until no improvement. Each generation consists of 10 ants, and we assume convergence. Since the tactic we use for dead ends and loops might make best routes unavailable, much more depends on the ants first, random route, if the route converges to a short route or an average one.
For the parameters of the graph, we decided to run ACO 10 times for each setting and plot the shortest route found among the ACO runs and average length of the shortest route. Additionally we also track the average number of generations passed until convergence as specified above.



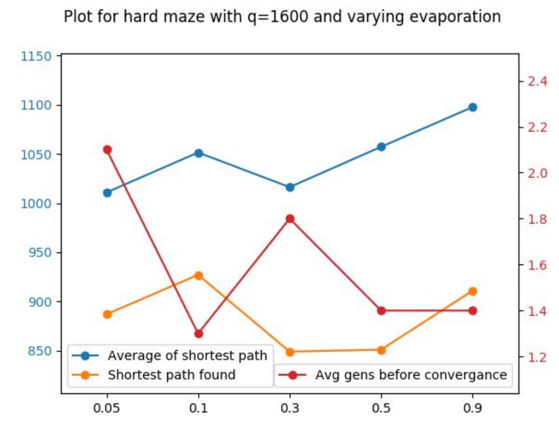


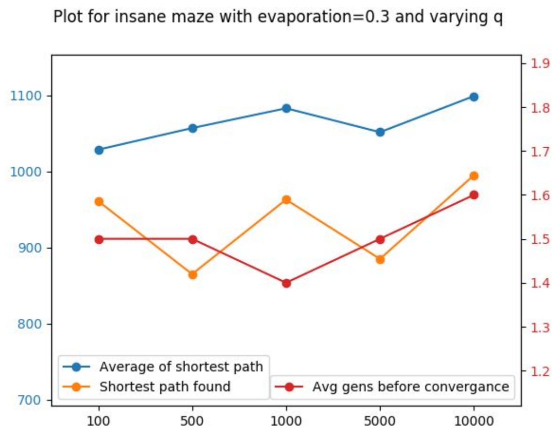
For the easy maze and q=1600, Evaporation value between 0.05 and 0.5 gave the same shortest route, but the average shortest and the fastest convergence appeared for value 0.1. The best q is around 500 for that evaporation value.



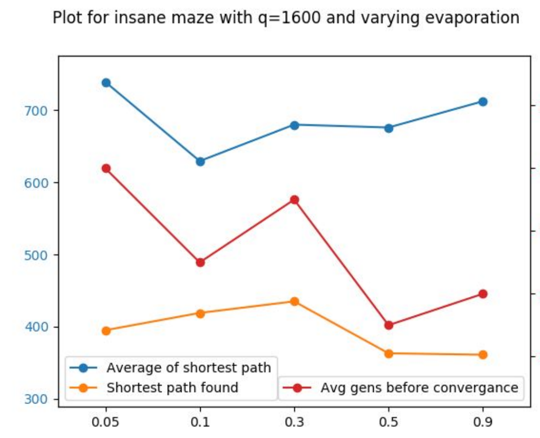


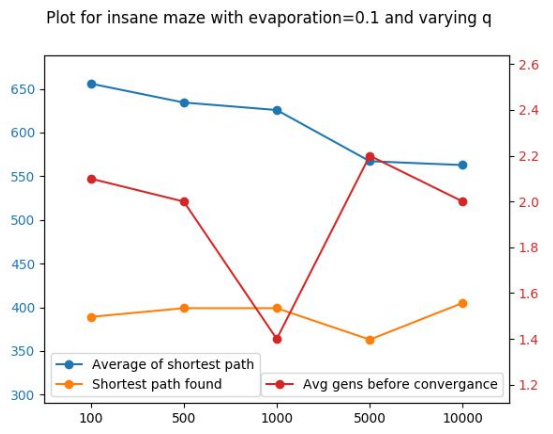
For the medium maze we decided that the best q is 300 and the best evaporation is 0.1





Note: the graph on the left has a mistake and is indeed for hard maze.
For the medium maze we decided that evaporation 0.3 and q 5000 are the best parameters





For the insane maze, the best parameters seem to be evaporation 0.1 and q 1000

7. It seems that the smaller the graph is, the smaller q can be. We also observe that the exact parameter values don't matter, what changes is the proportion between the starting value on all tiles, q and evaporation value. The bigger the evaporation value, the less initial value counts and more the last generation of ants counts.
Since our ants deal with open spaces (looping) in a way of building artificial walls, the parameters of evaporation don't really matter. We could have changed the way the ants treat tiles that lead to looping (e.g. imposing some amount of evaporation of the tiles that end up to looping, instead to setting them hard to 0) so the parameters would play a bigger part, but it's not feasible in the current time frame.

1.2 Problem Analysis

8. TSP: The regular travelling salesman problem states that, given a set of cities (nodes), and the distance between each pair of cities(nodes), the aim is to find the shortest path that visits each city(node) once and then returns to the starting city(node). Each city(node) is singularly connected to all the other cities(nodes).

9. In our situation, we do not have a graph with multiple nodes, we have a maze with a set of locations that need to be visited. We go from the starting point and visit each location from the given set of product locations and end at the destination point of the maze whereas in the original tsp, we would have to go back to the starting point.

10. Genetic algorithms are very good local search algorithms and are flexible. The environment and conditions in the travelling salesman problem may be unknown or could change over time (eg: a route may become unavailable) which is another reason why computational intelligence techniques are good solutions as they can adapt to the new changes while a dynamic programming solution would not be able to do this.

11. Genetic algorithms begin with a set of individuals known as a population and each individual from that population is characterized by a set of parameters known as the **genes** of the individual. Genes are combined into a string to form a **chromosome** and each bit of the

chromosome is a gene. A chromosome could represent the value of one or more parameters in the algorithm. In our case, the gene is a product that needs to be visited and a chromosome is a list of all the products that needs to be visited. We can encode these chromosomes using binary encoding where we have a string of 0s and 1s.

12. In our case, the fitness function can be the number of steps it takes to visit the products in the order given in a particular chromosome. From the population, the chromosome that requires the lowest number of steps to visit all its products will be the fittest chromosome.

13. The probability of a chromosome being selected depends on its fitness: the fittest individuals are selected to be the parents. For that we use a formula for the fitness ratio:

$$r(x_i) = \frac{f(x_i)}{\sum_j f(x_j)} \times 100\%$$

, where the numerator f is the fitness of chromosome i and the denominator is the sum of the fitnesses for all chromosomes.
From this we pick the two chromosomes with the lowest ratio as we are picking the chromosome that takes the least amount of steps.

14. Cross over: For the pair of parents picked, we use the 2 chromosomes to generate the offspring. We do this by picking a random crossover point and swap the bits after that point among the two chromosomes. This produces two offspring.
Mutation: For the new offspring, some of their genes can be subjected to a mutation with a predefined probability pm, (which should be small, otherwise it could lead to instability and longer search times). This means that some random products in the chromosomes need to swap positions.

15. In the TSP we need to visit each location only once and we can prevent points being visited twice by keeping track of the locations that have already been visited and making sure that they are not visited again.

16. The genetic algorithm will converge to a local minimum after a few generations and gets stuck there. In order to prevent this, we need to use mutations. Mutations will be applied to some individuals of a generation and they will usually make the result worse so they will not be selected for the next generation but sometimes the mutations cause an individual to get closer to a better local minima. The higher the mutation rate, the higher the chance of finding the global minima however, if the mutation rate is too high, the algorithm will not converge at all.

17. Elitism means that the most fit handful of individuals from the last generation are guaranteed a place in the next generation - generally without undergoing mutation. We have not used Elitism in our algorithm as we converge to a short path without having to do this.