Delft University of Technology

Computational Intelligence

CSE 2530

Assignment 2:

Evolutionary Computing

Authors:

Akdemir, Rauf

Farooq, Shah Muhammad

Khan, Zeeshan

Sahin, Tamer

Exercise 1

1) **Large open spaces**: Better to move towards goal than away from it
2) **Dead-ends**: Need to be able to revisit points otherwise cannot continue exploration and ants will be stuck in a dead-end.
3) **Unsolvable maze**: Make sure the solution does not keep on trying to solve an impossible problem.

Exercise 2

The equation for the amount of pheromones dropped is:

$$\Delta\tau^k = Q \times \frac{1}{L^k}$$

Where the amount of pheromones on path i is equal to the sum of the pheromones dropped by each ant k on that location. Here Q is a constant and L the length of path L.

We drop pheromones, because we want to know which paths have been visited by the ants of previous generations.

As the amount of pheromones are proportional to the length of the path, and the probability that a certain path is chosen is, amongst other things, dependent on the amount of pheromones dropped, the use of the amount of pheromones dropped will enable the ants finding the optimal path.

Exercise 3

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

Where rho is the evaporation constant, that affects how much of the pheromones that have been left behind on a route by ants in the previous generations, will evaporate before the next generation of ants starts searching for the optimal route.

Evaporation can be seen as an exploration mechanism. It delays faster convergence of all ants towards a suboptimal path, as the pheromones of the previous generations of ants have less impact on the search process of the new generation of ants. The decrease in pheromone intensity favors the exploration of different paths during the whole search process.

Exercise 4

```
AntColonyOptimization(maze, #ants, #generations, q, evaporation_rate)

final_route = empty list

while (#gen < generations):

        all_routes = empty list of lists

        curr_pos = position each ant in starting node

        for each ant:
                while(ant steps < max ant steps):
                        for all 4 directions,

                                        if in bounds,

                                        then calculate prob and move

                if curr_pos == finish_pos:
                        route = store paths taken

                        if |route| < |final_route|:
                                final_route = route
                        break

        all_routes.append(route)

        evaporate pheromones

        add previous pheromones in all_routes according to length of path

return final_route
```

Exercise 5

We make use of a set for visited coordinates to effectively remedy complex situations such as loops, open-areas and dead-ends.

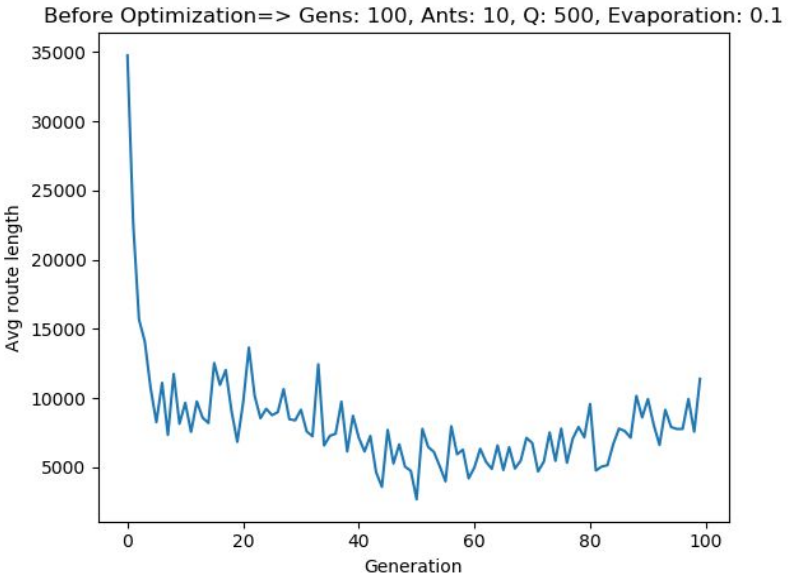
For loops: the ants will try to visit unvisited coordinates instead returning back and forth between points leading to shorter convergence times.

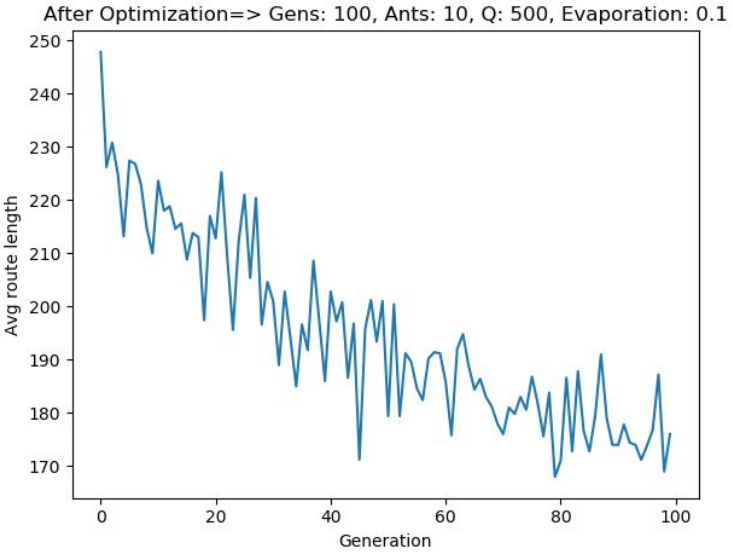
For dead-ends: if the ant reaches a dead-end where it cannot continue further it will need to visit an already visited cell, in this scenario it will randomly choose from all possible directions based on the calculated probability without looking at the already visited set.

We also take into account the inverse euclidean distance from the current position of the ant to the end position in the probability to encourage the ant to move in paths nearer to the goal.

For open areas: we keep an extra class member variable in Ant and Route to keep track of the visited positions and update them as we traverse further in the maze. If the calculated direction takes us to an already visited coordinate, we try to revert back in direction until we first see the visited coordinate and remove the steps in between.

Graphs below are tests for the medium maze



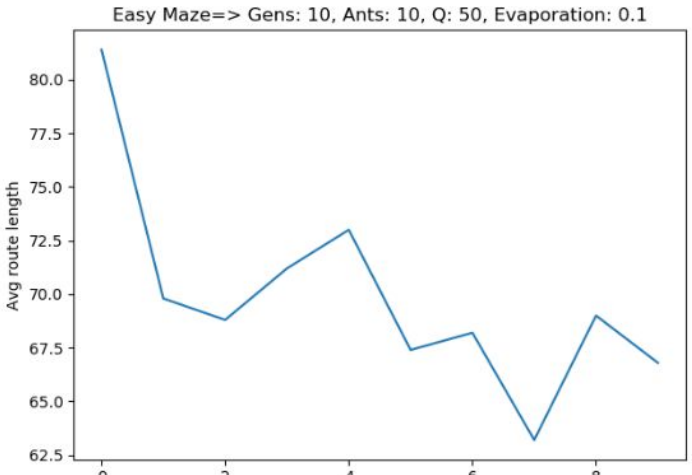


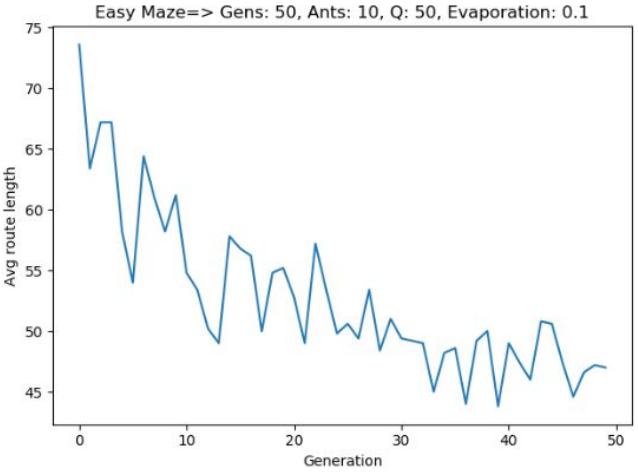
As seen in the figures above, after implementing the above mentioned optimizations we notice that the results converge quickly. The difference is quite significant already with the first generation since it avoids loops and getting lost in open areas enabling it to reach the global minimum.

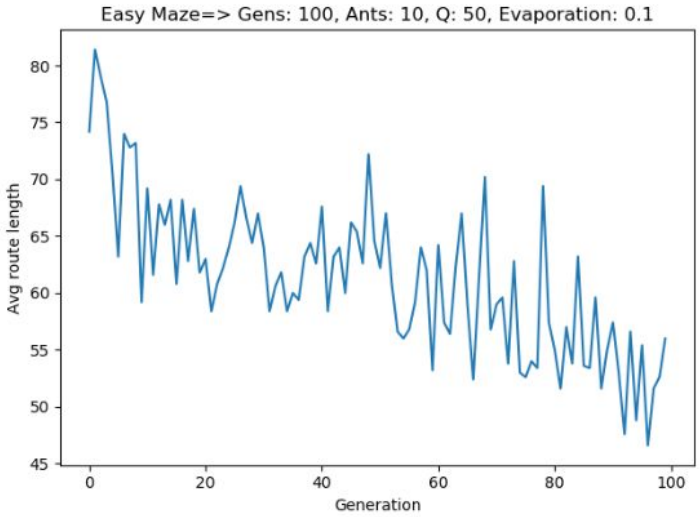
Exercise 6

We try to find the most optimal combination of hyperparameters by the trial-and-error approach. We found that generally Q normalizes the traversed route length for the next generation so it should be assigned a value near the expected route length.

Easy maze

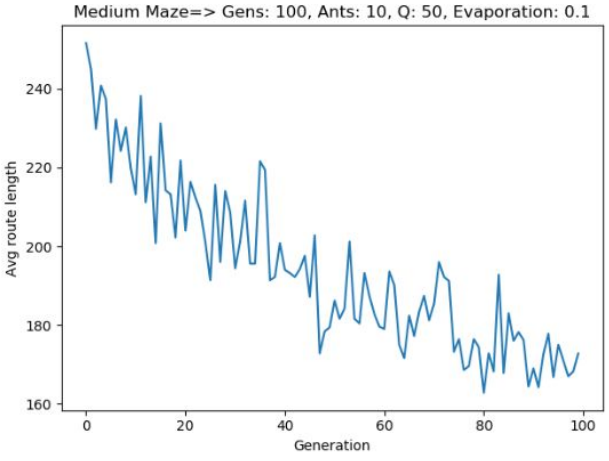


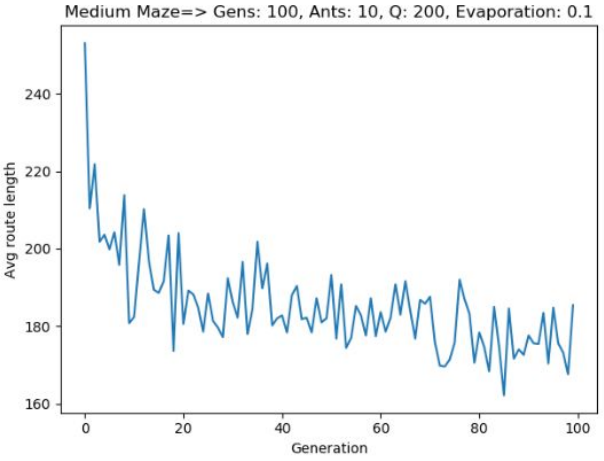


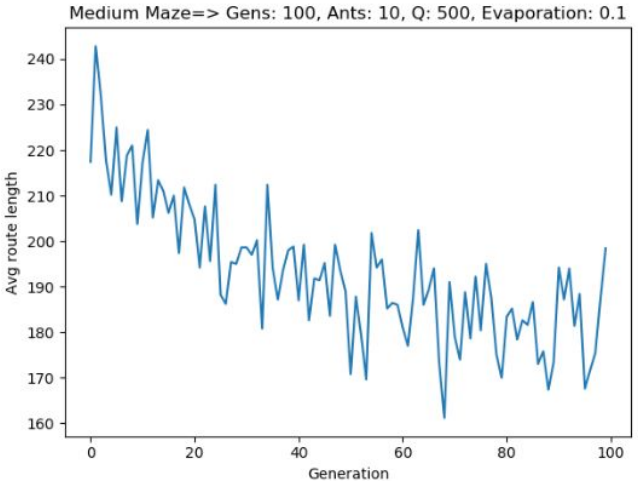


As seen in the figures above, we gradually increase the number of generations for the easy maze. We chose to adjust this parameter because the maze is quite simple relative to the other mazes and converges rather quickly. Larger number of generations makes the results get more noisy with higher variance in the average route length.

Medium Maze

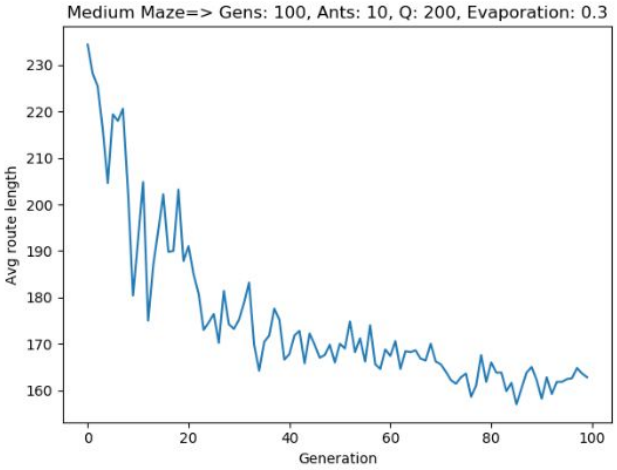


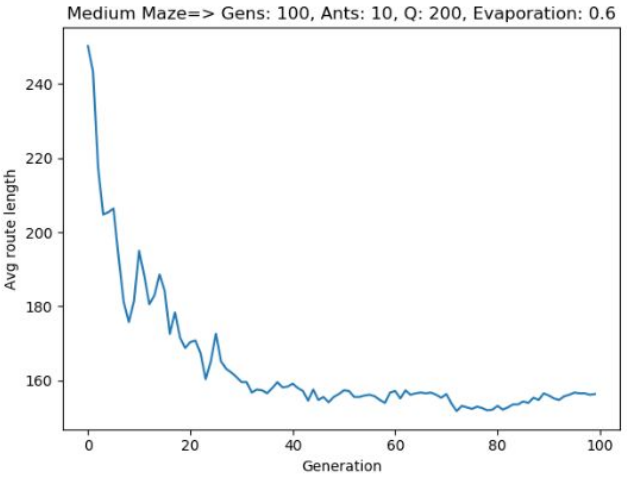


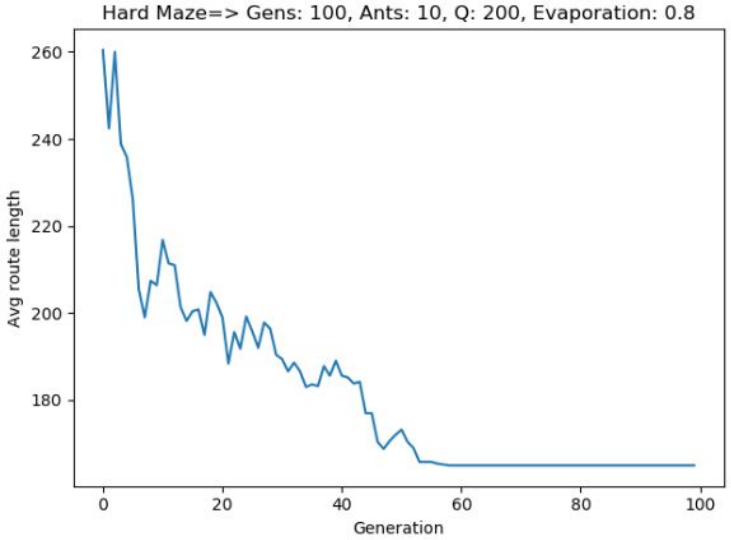


It takes more computation time with lower Q values than with higher ones leading to faster convergence times; however, this leads to big bounces from the minimas.

Evaporation

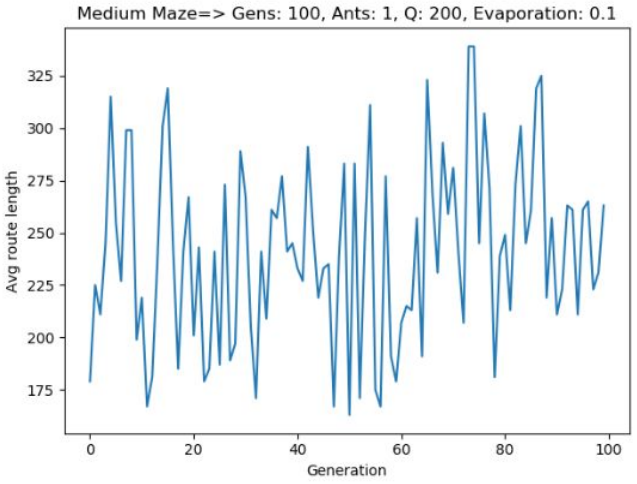


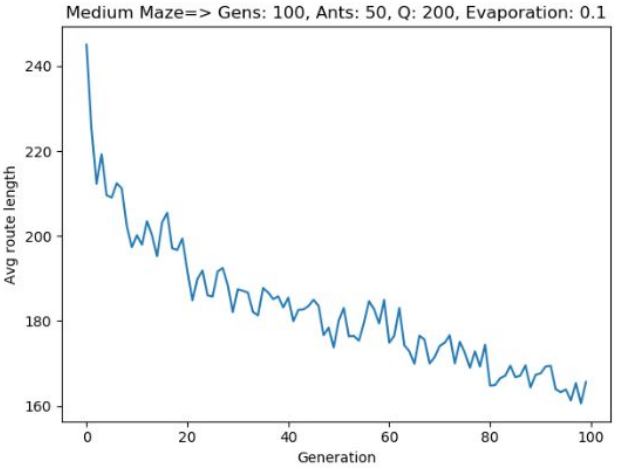




Higher evaporation rates tend to converge faster than lower ones. It is also noteworthy, that in higher evaporation rates the algorithm levels off and stays near a local optimum, rarely reaching the global optimum (135).

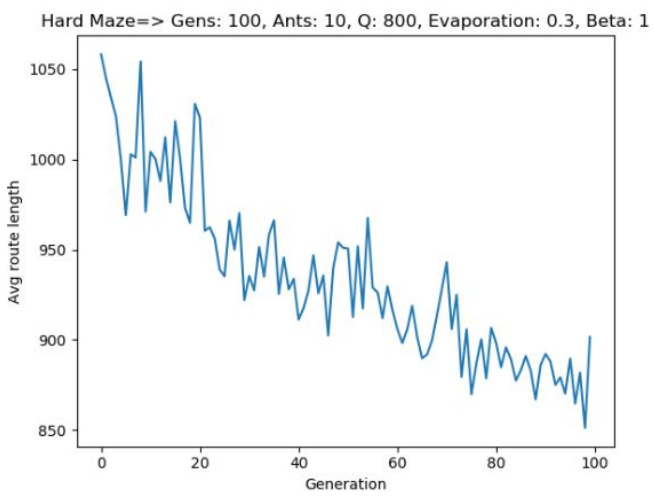
Ants

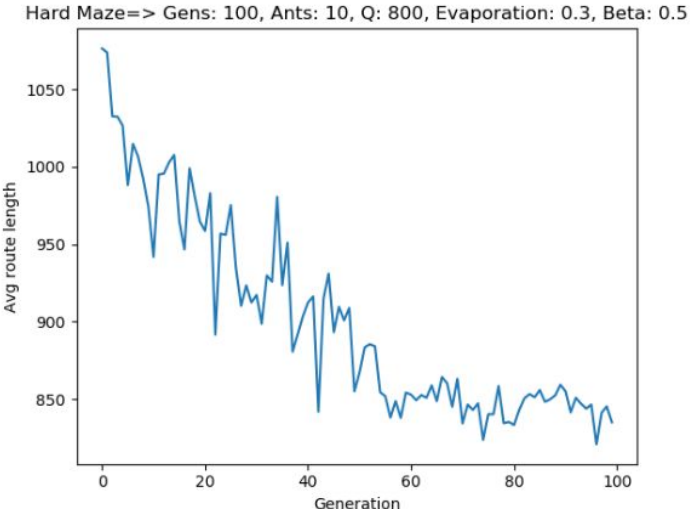




Lower numbers of ants per generation tend to have a shorter convergence time however this leads to the algorithm returning values from the local minimas. With the earlier results, we already see that 10 ants are adequate to reach near a global minimum, with higher numbers of ants per generation we tend to slowly but surely reach the global optimum as well.

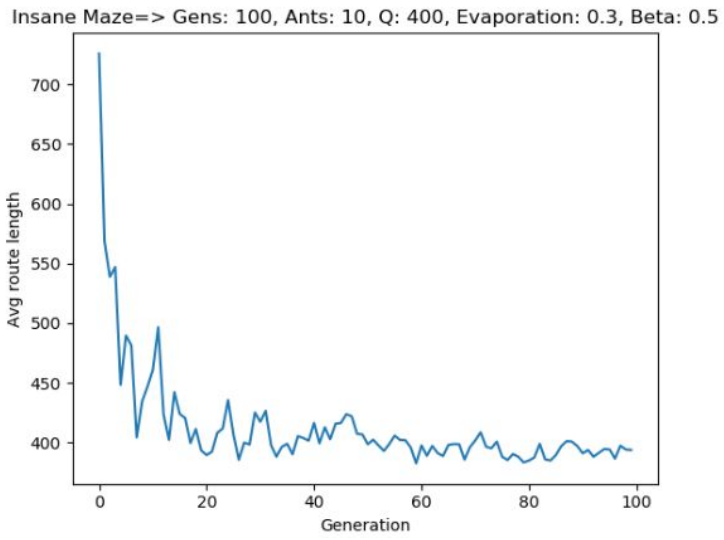
Hard maze

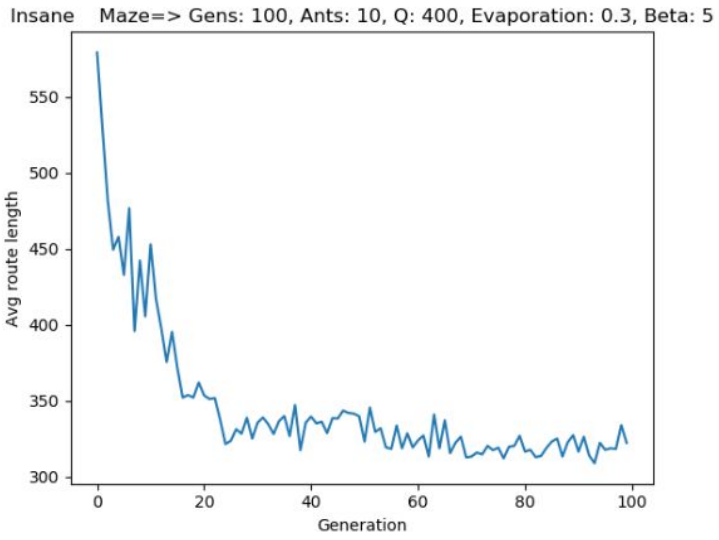




The hard maze takes longer to compute since there are more open areas. The figures above show that tweaking beta (path visibility factor) to higher values takes more time for convergence since it falls into some loops therefore a lower beta value is preferable which reaches the global minimum faster.

Insane maze





For the insane maze, the higher beta value makes it converge faster to the global minimum of 285 steps rather quickly.

Best parameters:

	Ants	Generations	Q	Evap	Alpha	Beta
Easy	10	50	50	0.1	1	1
Medium	10	100	200	0.1	1	1
Hard	10	100	800	0.3	1	0.5
Insane	10	100	400	0.3	1	5

Exercise 7

There is a direct relationship between the complexity of the maze and convergence time for the algorithm to find an optimal route. Increasing the number of generations and ants, increases the computational power requirement to execute the algorithm but makes it converge slower. The best idea we made use of was to adjust the q parameter (normalization factor) to the expected route length. This makes it converge quicker and closer to the global minima. Adjusting the beta also proved useful when more complex mazes were involved such as the insane and hard maze.

Exercise 8

The regular TSP problem arises when there are multiple locations (or nodes in a graph) that you want to visit. You visit all the nodes exactly once in a way such that the total distance covered in visiting these nodes is kept at a minimum.

Exercise 9

Our problem is not a singularly connected graph, instead it is a matrix that represents a maze through which one can move. The distance from one product to another is not known. The distances between these 2 points cannot be determined by calculating the euclidean distance. Instead we first need to calculate these distances using the Ant Colony Optimization Algorithm we constructed in part 1 of this assignment and then use these distances for our Genetic Algorithm to calculate the most efficient route possible for visiting all these products.

Exercise 10

TSPs have the nature of becoming very complex very fast as it has O(N!) complexity. This means brute forcing is in most cases not a feasible option. Computational intelligence needs to be applied to efficiently come up with a close to optimal solution.

Exercise 11

The genes represent products. The chromosomes are a list of these products, it contains all the products that we need to visit. The order of the elements inside the list represent the order in which the products are visited.

Exercise 12

The fitness function we used is the following:

$$F = 1 / (D + 1) \qquad (1)$$

where F is the fitness and D is the total distance of a route through all the products.This formula makes it such that a lower distance for a route leads to a higher fitness. We add 1 in the denominator just in case D evaluates to 0.

Exercise 13

For getting parents for the next generation we first calculate the fitness for each chromosome. This fitness is then normalized such that the sum of fitnesses of all the chromosomes in a generation becomes 1. The formula used for normalization is the following :

$$Normalized\ Fitness\ =\ F_{\mathrm{i}}\ /\ Sum(Fitness)$$

where F_{i} is the fitness of chromosome i in a generation and Sum(Fitness) is the total fitness of a generation.

We use this normalized-fitness to get parents for the next generation using the roulette selection method.

Exercise 14

Mutation: Mutation was carried out probabilistically for the algorithm. With a certain probability, the Algorithm randomly swaps the positions of 2 genes in a chromosome.

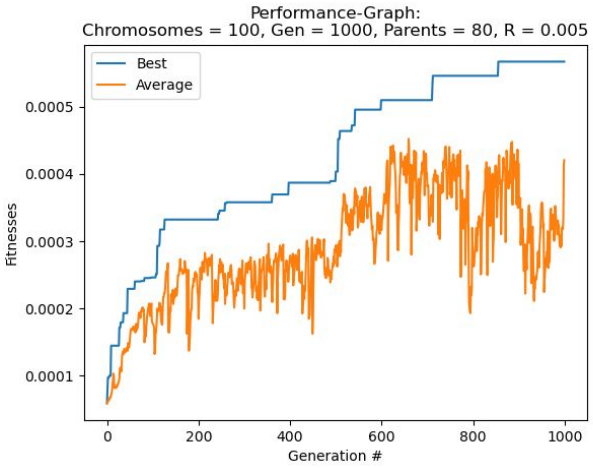
Cross-Over: For the cross-over we randomly select a subset of the first parent string and then fill the remainder of the route with the genes from the second parent in the order in which they appear, without duplicating any genes in the selected subset from the first parent. Below you can see a diagram illustrating how the crossover is performed by the algorithm to produce offspring:

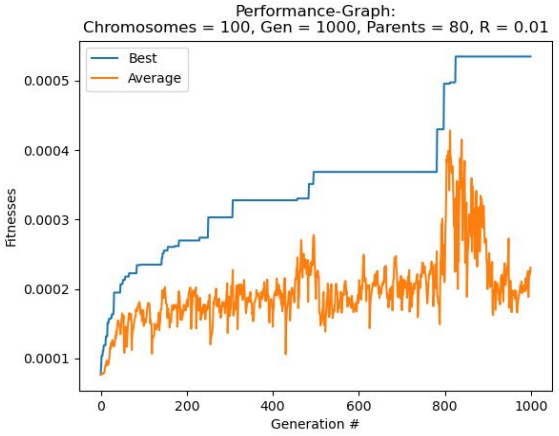
Exercise 15

The algorithm makes use of a list (chromosome) to store genes. During the initialization of the first generation, special care was taken to make sure there are no duplicates in the first batch of chromosomes. During the cross-over, the algorithm only adds the element in the offspring chromosome if it does not lead to the offspring having duplicate genes.

Exercise 16

We graphed the best-fitness and the average-fitness of each generation against the number of generations we used. We used different hyper-parameters to determine if the fitness converged to the same value. This gave us confidence that the distance we reached was in-fact a global minimum rather than a local minimum. As you can see in the performance graphs below, even after changing the mutation-rate(R) of the algorithm from 0.005 to 0.01, the Performance-graphs are pretty similar





Exercise 17

In a genetic algorithm we can apply elitism to produce the next generation. The goal of elitism is to keep the best chromosomes unchanged from the previous generations. For elitism we chose the parent chromosomes in a probabilistic way using the roulette wheel method. A percentage of these parent chromosomes were passed onto the next generation without any changes, while the rest of the remaining population for the next generation was populated by creating new chromosomes using the cross-over technique on these parents.