ECM3412 Nature-Inspired Computation

1 Introduction

An Ant Colony Optimisation (ACO) algorithm is one that is inspired by nature. The goal of an ACO algorithm is to mimic the behaviour of ants to solve a problem, whereby the fitness of a solution guides the next [1].

The problem consists of a number of locations, and a number of facilities. The distance between each location and the flow of students between each facility is recorded. The goal of this ACO algorithm is to place each facility in a certain location such that the sum of the distances between each facility multiplied by the flow of students between each facility is minimised.

The algorithm works by calculating the fitness for a given ant-generated path, and using the fitness to guide the next iteration of ants, with the help of pheromones.

In this report I will implement an ACO algorithm to solve this problem, and analyse my results to gain useful insight into the inner-workings of the algorithm.

2 Results

In the first stage of this investigation, I ran the algorithm with different parameters to try to understand how each parameter affects the outcome. There were four initial tests. For each test, the algorithm was run five times, with a specified termination criterion of ten thousand fitness evaluations. Both the best fitness from any of the runs and the average fitness per iteration were recorded.

m	**e**	Run 1	Run 2	Run 3	Run 4	Run 5	**Average**
100	0.9	5,672,302	5,639,624	5,671,806	5,725,244	5,668,658	5,675,526.8
100	0.5	5,658,408	5,676,316	5,642,598	5,680,914	5,652,136	5,662,074.4
10	0.9	5,693,824	5,671,464	5,658,320	5,690,286	5,682,588	5,679,296.4
10	0.5	5,695,750	5,676,570	5,687,702	5,688,810	5,653,614	5680489.2

Table 1: Results of each trial.

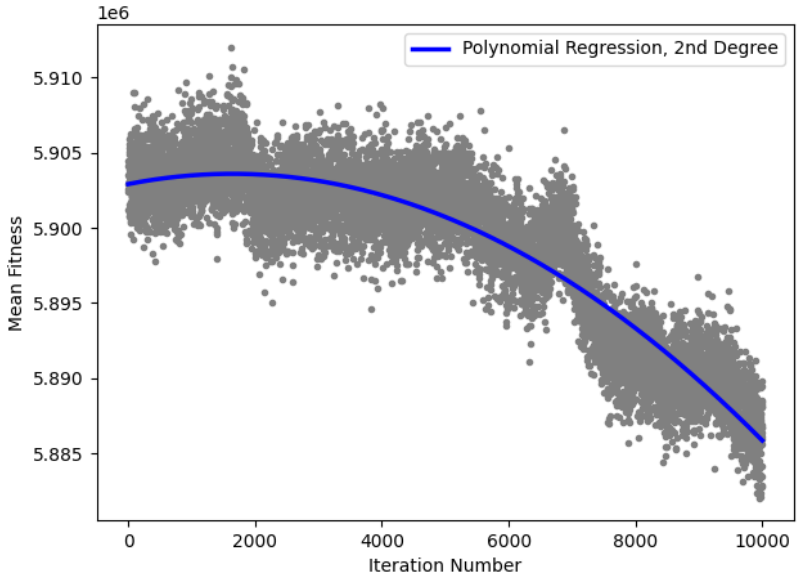
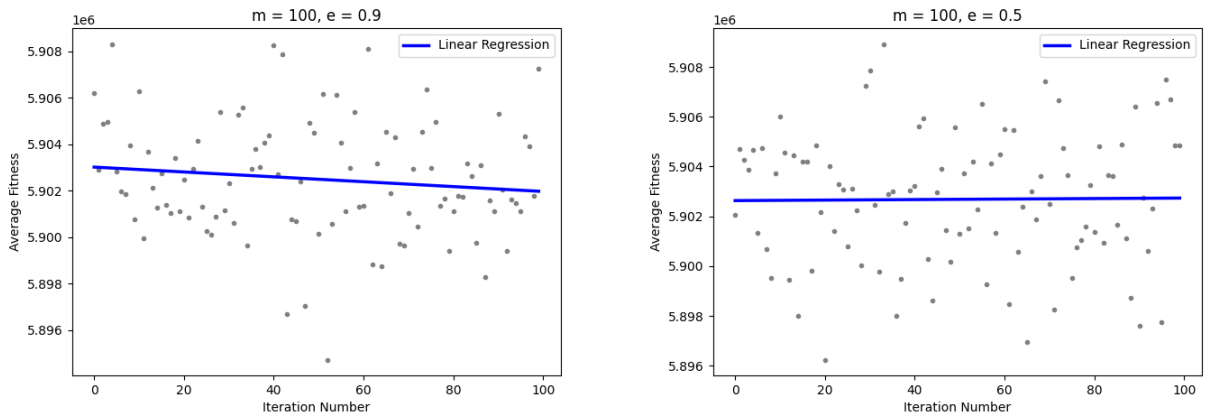
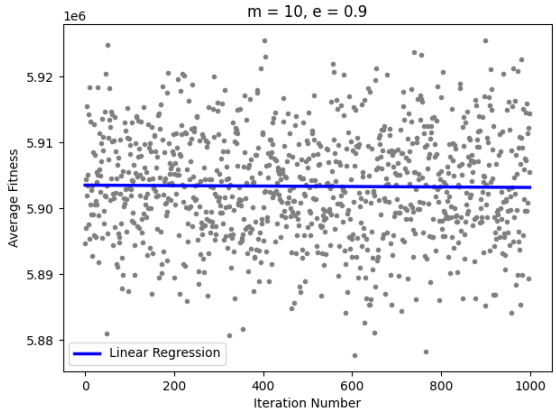


Figure 1: Trend of the program over 10,000 iterations.

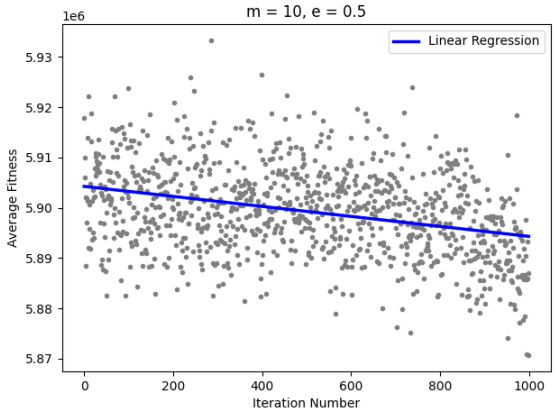


(a)

(b)



(c)



(d)

Figure 2: Change in average fitness over iterations, with different parameters.

3 Analysis

Which combination of parameters produces the best results?

In the initial stage of experimentation, the data in Table 1 shows that the combination of parameters which produces the best results is when $m = 100$ and $e = 0.9$. Despite not showing a strong negative correlation between the number of iterations and the average fitness per iteration in Figure 2, this combination proved to be optimal for achieving the best fitness.

What do you think is the reason for your findings in Question 1?

Since ten thousand fitness evaluations is a relatively small number, it would be expected that a trial with a higher evaporation rate would sooner find a better fitness. When the evaporation rate is higher, the algorithm is more likely to find a good solution quickly. This is demonstrated in Figure 2. In addition, having a higher number of ants per iteration allows the algorithm to further develop the pheromone links before moving to the next iteration, which has shown to reduce overall fitness.

How do each of the parameter settings influence the performance of the algorithm?

As described in the section above, there are two key features to changing the ACO algorithm's parameters:

- Decreasing e (increasing the evaporation rate) guides the algorithm to a good solution quickly, but not necessarily the best solution
- Increasing e (decreasing the evaporation rate) allows the algorithm to more slowly update the pheromones and allow more links to be explored for a longer time. This can result in the algorithm finding unexpected better solutions.
- Having an increased number of paths generated per iteration allows the algorithm to further mature all links in a graph. This reduces the speed at which iterations are computed, but increases the correctness and reliability of iterations. This will often lead to a better fitness overall.
- Decreasing the number of paths generated per iteration means that more iterations can be calculated quickly, but the graph of nodes is not explored as much per iteration. This will lead to the algorithm quickly finding a solution that may not be the global best.

Can you think of a local heuristic function to add?

I suggest that when an ant chooses the next node in its path, its choice should not be solely dependent on the pheromones laid on each link. I think that the distance from the current node to all potential next nodes should be considered when the ant chooses the next node in its path. In the current implementation of the algorithm, the distance between all nodes is not evaluated until all paths have been generated in an iteration. If the distances to all potential next nodes were implemented alongside the pheromones (perhaps *pheromones + 1/distance* for each node), the algorithm would soon prioritise placing facility pairs with a high flow closer together.

Can you think if any variation for this algorithm to improve your results? Explain your answer.

When an ant path's fitness is calculated, each node is considered independently. The fitness function takes each node in the path and sums the distance to all other nodes multiplied by the flow to and from all other nodes. While this is a suitable fitness function in theory, the specifics can prove rather computationally expensive. In order to preserve the relative fitness of a path in relation to all possible paths, while reducing the computational cost of calculating the fitness, I suggest that the fitness function should remember the nodes whose fitness has already been calculated, and ignore these nodes when calculating the fitness of the remaining nodes in the path. This will reduce the number of calculations performed when computing the fitness of a path, thereby improving the efficiency and reliability of the algorithm.

Do you think of any any other nature inspired algorithms that might have provided better results? Explain your answer.

An evolutionary algorithm (EA) might have provided better results. An EA is an algorithm where a generation of competing individuals gets a random assignment of a trait, and each individual's fitness is measured by this trait. The traits of the fittest individuals are combined and, with some random genetic mutation, inform the traits of the next generation [2]. For this

problem, the trait of an individual in an EA could be represented as its path, with the fitness function remaining the same as the original ACO implementation.

This type of algorithm may show improved results due to its difference in function. Since each generation of the algorithm is a mutated copy of a subsection of the last, it would come as no surprise to see the best fitness of each generation converge much faster to the theoretical best fitness than an ACO algorithm, where each new iteration is not so closely linked to the previous one.

References

[1] D. Angus *et al.*, "Ant colony optimisation: From biological inspiration to an algorithmic framework," *Technical Report at Swinburne University of Technology*, 2006.

[2] A. E. Eiben and J. E. Smith, "What Is an Evolutionary Algorithm?" in *Introduction to Evolutionary Computing*. Springer, 2015, pp. 25–48.