Ant Colony Optimisation Report

Method

The flow and distance matrices are created through the use of the numpy.loadtxt() function, to read the Uni50a.dat file and create a 50x50 array. In addition, the pheromone matrix is created by using np.random.rand(50,50) to create a 50 x 50 array of containing random numbers between 0 and 1. Importantly, the random generator is seeded with 5 different seeds per trial, but these are consistent between different configurations to allow for easier comparison due to removal of the bias from different starting pheromone matrices. The diagonals of the pheromone matrix are then changed to 0, and the matrix altered so that the values above the diagonal are equal to the corresponding values below the diagonal (e.g. [3,7] is equal to [7,3]).

The ants are created through the use of an ant class, containing information on the ant's path, cost, visited nodes and available nodes. The initial paths for these ants are created through the generate_path function. The start node is chosen randomly each time, to allow for maximum exploration of different paths, helping stop converging at a local minimum. A loop is then used to continuously find the next location based on the pheromone matrix, in which higher pheromone values result in a higher chance of being picked as the next location. This is done instead of simply picking the highest pheromone value to encourage exploration of the search space. Ants cannot visit the same node twice in a path, and this is maintained by checking the 'available' object variable. The costs for these paths are then calculated using the provided equation and stored in the 'cost' ant object variable.

The pheromone matrix is updated through both the evaporate_pheromone() and deposit_pheromone() functions. Evaporate pheromone simply removes a given percent of the pheromone. Meanwhile, deposit pheromone adds pheromone to the paths used by 1/cost of the path created.

Finally, the best path generated based on the lowest cost per saved for each trial and used as the final result.

Results

In running the algorithm with evaporation rates of 0.5 and 0.9, as well as number of ants of 10 and 100, we get the results shown within figure 1.

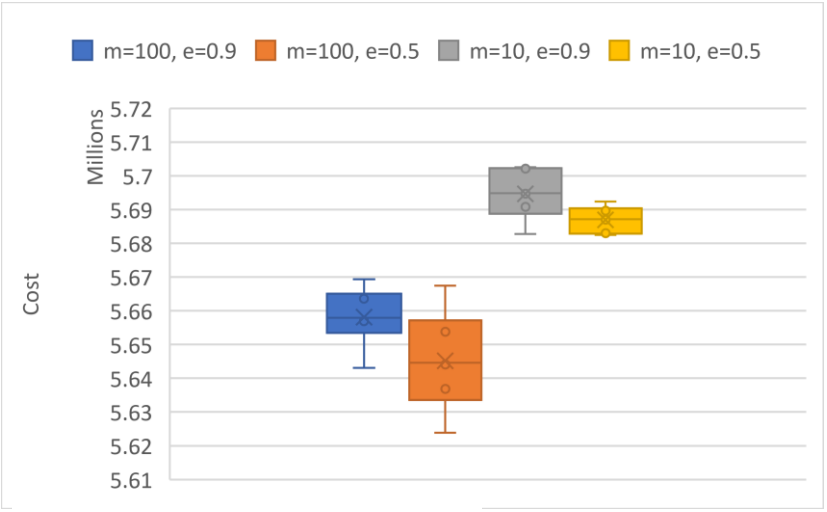

Figure 1: results from all trials

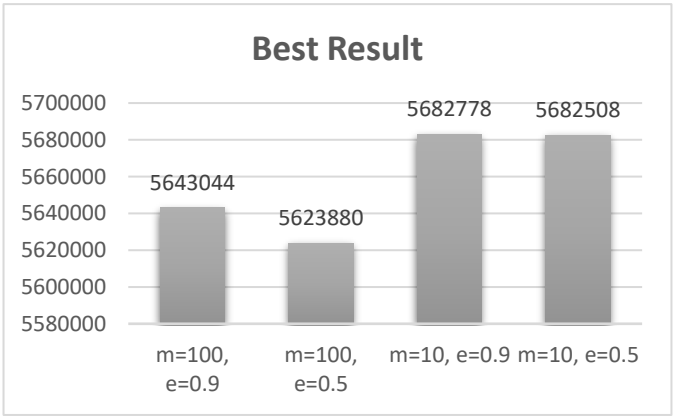

Figure 2: best result for each trial

These results show that 100 ants result in a lower cost from the best generated path in 10,000 evaluations compared to 10 ants. An evaporation rate of 0.5 also performs better than a rate of 0.9. In addition, it is worth noting that 100 ants have a wider spread of results between runs when compared to the trials with 10 ants. Furthermore, these conclusions are further shown in figure 2, which shows the best result (lowest cost) generated for each trial, with m=100 and e=0.5 performing best once again.

1. Which combination of parameters produces the best results?

The results shown in figures 1 and 2 show that 100 ants and an evaporation rate of 0.5 gave the best results, producing the lowest cost for the generated path. Each configuration was run 5 times, using 5 different seeds for the random number generator to create the initial pheromone matrix, to allow for more consistency between the different trials.

Between each different configuration the lowest average cost is generated with m = 100 an e = 0.5, followed by m=100 and e=0.9. This shows that the main factor that helps generate more efficient paths is the number of ants, with 100 ants leading to a better paths. Furthermore, the evaporation rate also has an effect on the paths generated, with an evaporation rate of 0.5 performing better than a rate of 0.9.

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

These results are as expected, as a larger number of ants allows for more exploration of the search space due to more paths being generated before updating the pheromone matrix. With 100 ants, 100 different paths are generated before the matrix is updated, allowing for a more varied pheromone matrix due to the random start node each iteration. This results in more exploration of the search space, helping stop convergence in a local minimum. In contrast, 10 ants means that the pheromone matrix is updated after only 10 paths are created, so the pheromone is only deposited along a small number of paths. The pheromone matrix therefore becomes heavily weighted to these 10 paths, and these paths become increasingly likely to be generated again in the next iteration, causing convergence at a local minimum. Whereas, with 100 ants a larger part of the matrix maintains higher values due to more depositing of pheromones over a larger number of paths. Therefore, the algorithm does not get stuck in local minimums as easily, as shown in [1], in which 100 ants outperformed 10 and 200 ants for the same reasons described.

In addition, an evaporation rate of 0.5 outperforming 0.9 is also as expected. This is because an evaporation rate of 0.5 results in slower reduction of pheromone in paths that are not used, and less weight being placed on paths found. This allows for more exploration of new paths, helping to stop an early convergence at a local minimum. An evaporation rate of 0.9 results in the pheromone matrix tending to zero in the unused paths as the pheromones decrease too much between iterations, and only the used paths have pheromone values of any significant value. These paths are therefore picked again in the next iteration, leading to convergence at a local minimum, as shown in [1], in which 0.20 outperformed 0.05 and 0.95 for the same reasons described.

Therefore, values of m=100 and e=0.5 lead to the best paths generated with the lowest cost, as more ants and the lower evaporation rate maintain a more varied pheromone matrix, helping to stop convergence at a local minimum. In addition, predetermining the start node was also tested, however this let to stagnation in the generated paths with poor results due to not being able to find the optimum path from the start node.

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

The key factor that influences the performance of the algorithm is the number of ants. This can be seen in figures 1 and 2, in which it is clear to see that 100 ants significantly outperformed 10 ants, as more ants allows for greater exploration of the search space. This is important, as the pheromone matrix is randomly generated, so the initial best path is not the same as the optimal best path. When using a small number of ants, the pheromone matrix becomes heavily weighted to these 10 paths from the random matrix, leading to convergence at a local minimum. More ants help avoid this, leading to a better solution. However, too many ants can also have a negative effect, causing longer convergence times due to too many paths being created before updating the pheromone values. In further experimentation I found no improvement in results with 200 ants, and a significant decrease in performance with 300 ants, with an average cost of 5707972 with an evaporation rate of 0.5.

In addition, an evaporation rate of 0.5 outperforms 0.9, due to 0.9 changing the pheromone matrix too rapidly, causing early convergence at a local minimum. However, pheromone values at both high and low extremes can be equally as detrimental to the performance of the algorithm. Values too low don't encourage continued exploration of the best paths. Whereas values too high put too much weight onto previous paths, causing convergence at a local minimum. Therefore, a value of 0.5 is a good option for both favouring previous paths and encouraging exploration.

4. Can you think of a local heuristic function to add?

In the experiment, the next location in a path is generated by through the use of probability based on the pheromone value between locations. This is done by summing the pheromone values for all available locations and creating a probability for each available location through:

(sum of available)/(pheromone value for location)

This approach works well, but a local heuristic would likely perform better through the use of more information (e.g., flow and distance) to decide the next location in a path. A potential heuristic that could be used is as follows:

$$\eta_{i,j} = \frac{1}{D[i][j] * F\big[p[i]\big]\big[p[j]\big]}$$

This can be used to generate a probability for each available location through the equation:

$$P_{i,j} = \frac{[pheromone_{i,j}]^{\alpha} * [\eta_{i,j}]^{\beta}}{\sum_{h \in available}[pheromone_{i,h}]^{\alpha} * [\eta_{i,h}]^{\beta}}$$

This probability equation favours locations with smaller distance * flow to the next location, as well as favouring higher pheromone values. Alpha and beta are hyperparameters, with higher values of alpha resulting in a more greedy algorithm, favouring the lowest distance * flow values. Whereas higher beta values favour the pheromone values in the matrix to determine the next location. These hyperparameters can be adjusted to help the algorithm avoid falling into local minimums to obtain a better solution.

In addition, simply choosing the highest pheromone as the next location in the path was tested. However, this led to convergence at a poor results due to always choosing the same path for each ant when compared to the probability method described.

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

One variant of the typical ACO algorithm that could be used is the Cunning Ant System (CAS), as described in [2] and [3]. The main idea here is to use parts of already existing solutions from previous iterations, with the remainder of the solution being generated through the normal ACO method. This is done through the use of a donor ant (d-ant) and a borrowing ant (c-ant). When c-ant borrows part of a solution from d-ant, the c-ant and d-ant are compared, with the best continuing into the next iteration. This allows for the best solutions to remain between iterations, and new solutions to be generated from these solutions. The number of sampling and borrowing ants is a hyperparameter to be tuned to encourage or discourage exploration of different paths or narrow in on the best solutions. This algorithm has had a lot of success, outperforming other variants such as MMAS and ACS, which both in turn outperform traditional ACO, as shown in [2]. Therefore, its implementation into this project would most likely result in better performance and solutions due to its ability to use previous knowledge in the creation of new solutions.

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

An evolutionary algorithm is one example of a nature inspired algorithm that could have been used in place of ant colony optimisation. Evolutionary algorithms take inspiration from the theory of evolution by generating a population, using crossover and mutation functions to generate new solutions based on the best in the population. The crossover function is a function that combines two or more parent solutions to generate new child solutions, with the idea that by combining features of the best solutions, new and better solutions can be created. Mutation function is a function used on a small proportion of solutions to slightly change an existing solution. By combining these functions, a population of solutions is altered over time, taking the best solutions in a population, and using these functions to generate new solutions. If these generated solutions are better than the parent solutions, they may replace the parent solutions in the population. This leads to a population that becomes better over time, with better fitness values for the solutions in the population.

Evolutionary algorithms have been very successful in quickly converging to a near optimal solution, which could be very useful in this project to find a good solution quickly. The different approaches taken by ACO and evolutionary algorithms may result in different solutions, due to evolutionary algorithms using previous solutions to create new ones, rather than using a global matrix. It is not possible to know which approach will work better for this particular project. However, one key advantage of the evolutionary algorithm is the use of a mutation function that allows for the escaping of local minimums, which ACO algorithms struggle to do. Evolutionary algorithms may therefore perform better than ACO.

In addition, algorithms exist that combine these two methods, allowing for the benefits of both methods to be exploited, giving a better chance of finding the global optimum. For example, the Cooperative Genetic Ant System described in [3] and [4], in which the initial solutions are generated by ACO. A second generation of solutions is then generated by both ACO and an evolutionary algorithm, with the best solutions being chosen and pheromone values updated accordingly. An algorithm such as this would likely give better results than a regular ACO due to using the benefits of both methods.

References

[1] T. M. Miranda et al., “On the Influence of the Variation Parameters of the Ant Colony Optimization on the Dispatch of Road Crews of Electricity Utility,” Energy Procedia, vol. 107, pp. 33–40, Feb. 2017, doi: 10.1016/j.egypro.2016.12.126.

[2] S. Tsutsui, “cAS: Ant Colony Optimization with Cunning Ants” Parallel Problem Solving from Nature - PPSN IX, pp. 162–171, 2006

[3] N. Sakthipriya & , T. Kalaipriyan , Variants of Ant Colony Optimization - A State of an Art. Indian Journal of Science and Technology, 2015

[4] H. Ismkhan, “Accelerating the ANT Colony Optimization By Smart ANTs, Using Genetic Operator,” arxiv.org, Nov. 2014