Project Report

Manuela Bergau
s4543645,

Stergios Morakis
s1047752

June 2020

1 Introduction

1.1 Travelling Salesman Problem & Ant Colony Optimisation

The Travelling Salesman Problem (TSP) was first formulated in 1930 and is known to be an NP-hard problem. Nevertheless, there are many algorithms that may provide solutions given sufficient amount of time but as the number of cities increases, the problem gets even more convoluted. For instance, the Nearest Neighbour algorithm is considered the simplest one, focusing on the nearest unvisited city at each iteration step [1]. In contrast, Genetic Algorithms have been particularly successful on finding the TSP global optimal by attempting to optimise the fitness function of each so-called individual, with various techniques inspired by nature [2]. The last group of dynamic algorithms used in TSP consists of Greedy Heuristic Algorithms, which focus on optimising each solution's best local optimal and via that process, converge to the global optimal [3].

The Ant Colony Optimisation (ACO) is a meta-heuristic algorithm that has been successfully used on complex TSP instances multiple times in the past. There are different variants to this algorithm available, such as Ant Colony [4], Ant Colony System [5] and Max-Min Ant Colony [6], each addressing different aspects of the algorithm's properties. Through this project we aim to investigate the performance of the mentioned variants of the Ant Colony Optimisation [7] in-depth, especially in terms of trade-off between exploitation and exploration, convergence, computational costs and their retrieved solutions.

1.2 Dataset

In 2009, the University of Waterloo in Canada published a TSP instance with 100.000 cities that reflects a representation of Leonardo da Vinci's Mona Lisa when observed from the right angle [8]. The creator of the dataset, Robert Bosch, uses optimisation to create visual art work, the so-called TSP art. To create such TSP art, the paper [9] by C. S. Kaplan and R. Bosch describes how to turn a black and white image into a TSP problem, while various methods are investigated on how to sample cities based on the picture's brightness. When the shortest route is found by solving the TSP instance, the result is resembling a line drawing of the used picture. Mona Lisa is an elegant example of that paper, that due to its complexity the optimal solution has not yet been uncovered. The current best tour has a length of 5,757,191, even though theoretical evidence has been given that the optimal length should be 5,757,084. An optimal solution to the 100,000-city Mona Lisa instance would set a new world record for the TSP. In this project we used Ant Colony Optimisation algorithms on this complex instance of TSP and more extensively on a smaller part of it. Figure 1 displays the plotted 100.000 coordinates of the data points, with opacity set to 0.2 due to the density. Undoubtedly, applying ACO on this instance of TSP has been an interesting experience for us, but quite challenging as well. Although we were not able to retrieve the results we sought due to hardware and time limitations, we applied our implementation on a much smaller sample of the data set consisting of 39 cities, the nose of Mona Lisa.

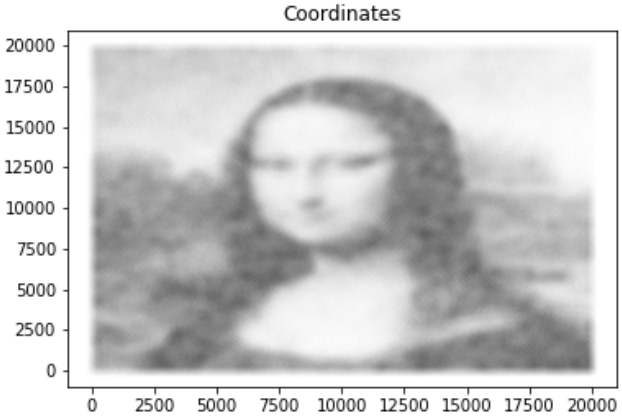

Figure 1: Datapoints of the Mona Lisa TSP problem

2 Background Literature

Ant colony optimisation was first introduced in 1992 as a swarm intelligence algorithm with metaheuristic properties, in the PhD thesis of Marco Dorigo [10]. That paper described a novel methodology for determining the optimal path in a graph, inspired by the behaviour of ants in the process of seeking food. A wide variety of optimisation techniques were applied on top of that original idea and resulted in many different variations.

The very first of them is Ant Colony System (ACS) which modified the original idea in very interesting ways. One modification of the colony system is the enhanced exploitation for the agents during the edge selection step. Secondly, the agents can apply a local pheromone rule on their solutions while building them, influencing future ants passing by. Lastly, the agent that achieved the best score is the only one affecting the global pheromone levels by applying a global pheromone rule on top of the local ones [5].

Next, Elitist Ant System introduced ways of preventing stagnation by allowing only the ant that achieved the best solution in each generation affect the global pheromone levels. Similarly, Rank-based ant system followed the same principles but allows all ants to affect the pheromone levels, instead of just the best one [11]. This is done by first ranking them based on their achieved scores and by assigning weights based on their rank.

Later on, Min-Max system introduced the concept of using minimum and maximum values to control the pheromone amounts on each trail, by not allowing an edge to exceed the maximum or get lower than the minimum parameter [6]. This setting has been quite popular especially in large TSP problems as the algorithm never classifies a path as irrelevant and thus prevents stagnation. A non-exhaustive list of other algorithms worth mentioning includes Continuous Orthogonal Ant Colony and Recursive Ant Colony Optimisation.

Furthermore, the paper [7] describes various methods on how the ant colony optimisation can be applied to the generalised TSP (GTSP) problem domain. In contrast to the TSP problem we are discussing in this report, the cities in GTSP are divided into groups. The shortest route has to visit exactly one city of each group.

However, ACO has a problem when reaching the global optimal solutions for TSPs, and the algorithmic performance of ACO tends to deteriorate significantly as the problem size increases. In the proposed modification of [12], adaptive tour construction and pheromone updating strategies are embedded into the conventional Ant System (AS), to achieve better balance between intensification and diversification in the search process.

3 Methods

We implemented Ant Colony [4], Ant Colony System [5] and Max-Min Ant Colony [6] using Python. We used a grid search to test different parameter values. A list of the tested parameters using a grid search can be found in appendix A. In the following section we will describe each of the algorithms along with some implementation details.

3.1 Ant System

The Ant System algorithm was developed in 1991 and got published in 1996 [13]. It is inspired by the way ants are dealing with obstacles when they are transporting food back to their home. Ants are leaving a trail of pheromone on the route they are taking. The stronger a pheromone on a trail is the more likely the next ant is picking that route as well.

The algorithm consists of two steps. First the ants construct a possible solution, picking edges with a probability based on the pheromone level. Then the solutions are compared against each other and the edges pheromone levels are updated based on the number of good and bad solutions that contain this edge.

Algorithm 1: Ant Colony Optimisation Meta Heuristic

```
while generations limit not exceeded do
    generateSolutions();
    LocalPheoromoneUpdate() (Optional);
    pheromoneUpdate();
end
```

3.2 Ant Colony System

Ant colony system is an improvement of the original ant system [5]. It has a local and a global pheromone update. The global update is based on the best solution only. Furthermore ants are more likely to select edges with a large amount of pheromone.

3.3 Min-Max Ant System

The Min-Max variant of the ant colony optimisation algorithm is designed to avoid stagnation [6]. It has two parameters (min and max) to control the minimal and maximal possible pheromone level on an edge. All edges start with the maximal pheromone level and the pheromone is set back to the maximum is stagnation is reached.

In our implementation we aimed for the large dataset, we set the minimum to $(\frac{1}{\#cities})^2$ and the maximum to $\frac{1}{\#cities}$. Testing those parameters on the large set was difficult and thus we evaluated their influence on a much smaller set using grid search.

3.4 Parameters

We use use various parameters that all algorithms have in common and some that are algorithm specific.
The general parameters are:

- α: the pheromone decay parameter controls how strong the ant is attracted by the pheromone.
- β: is a control parameter to determine the importance of pheromone and distance.
- ρ: this parameter controls how long the pheromone stays on the edge.

The specific parameters for Ant Colony System are:

- φ: this parameter is similar to ρ but is used for the local pheromone update.
- q_0: with this value we can control the exploration-exploitation ratio. with probability q_0 the best known edge is chosen (exploitation) otherwise other edges are explored.

The specific parameters for Min-Max Ant System are:

- *minimum*: The minimal pheromone value.
- *maximum*: The maximum pheromone value. The edges are initialised with this value and are reset to this value if stagnation is reached.

3.5 Additional techniques

The Travelling Salesman Problem is classified as an NP-hard problem. Consequently, due to the large number of cities we are dealing with, it was necessary to apply an ad hoc approach on top of our implementation to reduce the algorithm's run-time, as the simplest one for example, Ant System, was estimated to have a total run-time of approximately 2 days. In addition, we introduced our own version of 2-opt local search in our system which offered a positive effect on the results.

- Closest Neighbours

  Edge selection has always been a bottleneck in ACO variants. Especially in our case, where we have to deal with a large NxN matrix, the number of choices to be taken into consideration for each iteration is overwhelming. We believe that in a large and dense graph, such as the instance we are dealing with in this project, applying the edge selection process on a limited amount of nearest neighbours not yet visited, will drastically improve the overall performance while not affecting substantially the convergence to the global optimal. After some experimentation, we agreed to adjust edge selection on the 10% closest cities not belonging in the tabu list.

- 2-opt Local Search

  We were quite sceptical regarding the use of such simple yet effective local search algorithm, due to the size of our data. The question was whether further improving an ant's existing solution was worth the tough challenge of identifying an adjacent route that would ultimately guide us to the 2-optimal tour. Therefore, we applied it without necessarily depending on it, as best described in Algorithm 2, by setting a maximum number of trials. It is worth reminding the reader, that the distance matrix was computed in the beginning and thus, evaluating and comparing the new tour's euclidean distance is considered a trivial task. The optimal number of trials is worth investigating further, but due to time limitations we are unable to showcase results at this point on the large dataset.

Algorithm 2: Ad hoc Local Search

```
parameters = best_tour, best_cost;
trials = 10;
while trials > 0 do
    i, j = random integers (restricted to the number of cities);
    adjacent_tour = best_tour[start:i] + reverse(best_tour[i:j])+best_tour[j:end]);
    adjacent_cost = compute_cost(adjacent_tour);
    if best_cost > adjacent_cost then
        best_tour = adjacent_tour;
        best_cost = adjacent_cost;
    else
        trials = trials - 1;
    end
end
```

4 Evaluation

We ran the three algorithms for all possible combinations of parameters shown in appendix A. For the computations we made use of Ponyland servers in Radboud University. For each output, we then computed the mean average score to better illustrate our findings, defined as the averaged computed distances for all ants in a generation, averaged over the total number of generations for that specific setting of parameters.

4.1 Results

In the following sections we want to have a closer look at the influence of different parameters on the output of the algorithm. Note that it is impossible to infer direct causality from our results. We divide our analysis in two steps: a comparison of the parameters all algorithms have in common and an analysis of the algorithm specific parameters. We will focus on the most interesting parameters as it will be too much otherwise. Figure 2 shows the result of the best run of each algorithm together with the parameters used.

In general ACS has a lot more parameters than the other two algorithms, that also explains the large difference in computation time in figure 3. Due to the long run-time, we could not test more different values, which in some cases results in little data to analyse.

We are also interested in the effect of the closest neighbour selection.

	AS	ACS	MMAS
best	3470.51	3455.64	3458.4
α	1	1	0.2
β	2	1.5	2.5
ρ	0.6	0.2	0.2
updates	8	9	19
φ	-	0.4	-
max	-	-	0.9
min	-	-	0
average mean cost			

Figure 2: Best result of each algorithm with the used parameters

4.2 Closest Neighbours

Figure 3 shows a slight improvement in total run-time for the algorithms, even for the small set of 39 cities when restricting the possible edges to the 10% closest cities. A big advantage of this method is, that the list can be pre-computed and computations are only done once.

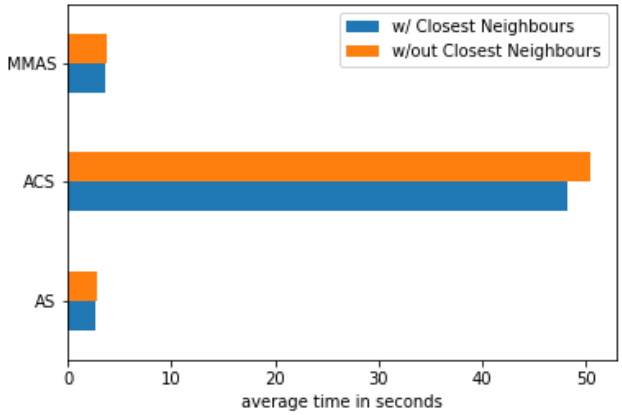

Figure 3: Computational times

4.3 Pheromone decay α

Figure 4 shows the different alpha values. It is interesting to note that the min-max algorithm is not that dependent on this parameter as the other two algorithms.

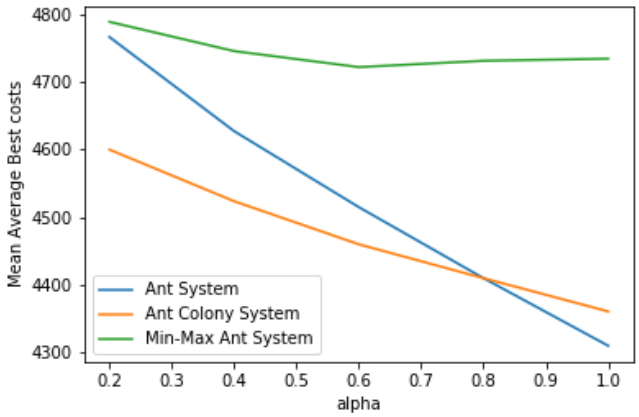

Figure 4: the influence of α on the mean average score

4.4 Pheromone residual coefficient ρ

As we can see in figure 5 all algorithms behave similar when ρ is increased.

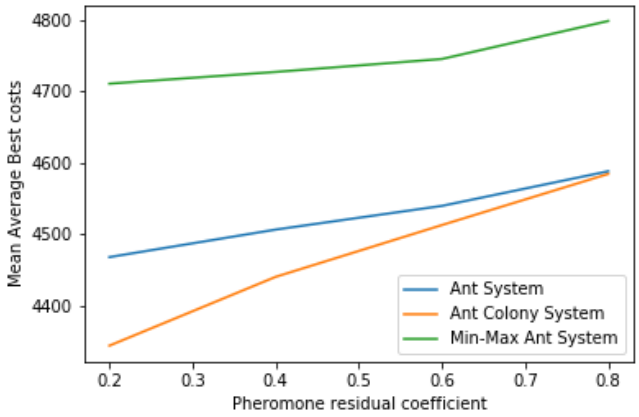

Figure 5: the influence of ρ on the mean average score

4.5 Minimum and Maximum

The min-max ant system algorithm gives us the possibility to adjust the maximal and minimal possible pheromone value. Our initial idea was, that a large difference between the two values would result in a better output. That does not seem to be the case. What we can well see is, that a small minimum value gives the best results independent of the maximum value used.

min	max	average mean
0.0	0.6	**4163.35**
	0.8	4180.28
	0.9	4177.66
	1.0	4177.6
0.1	0.6	5029.56
	0.8	5032.24
	0.9	**5028.17**
	1.0	5030.91
0.2	0.6	5032.00
	0.8	**5029.92**
	0.9	5030.57
	1.0	5030.69

Figure 6: the parameters min and max together with the mean average score

4.6 Ant Colony System

4.6.1 q_0

The values we used for the q_0 parameter indicate a linear dependence between the value and the score (see figure 7). Our problem instance has a dense city spread, so it is logical that picking the closer cities results in a better overall score. That was the initial idea to restrict the list of possible cities to the 10% closest once. q_0 is the probability to select the best found city (exploitation). It is also interesting to note the difference of the two variants in figure 7.

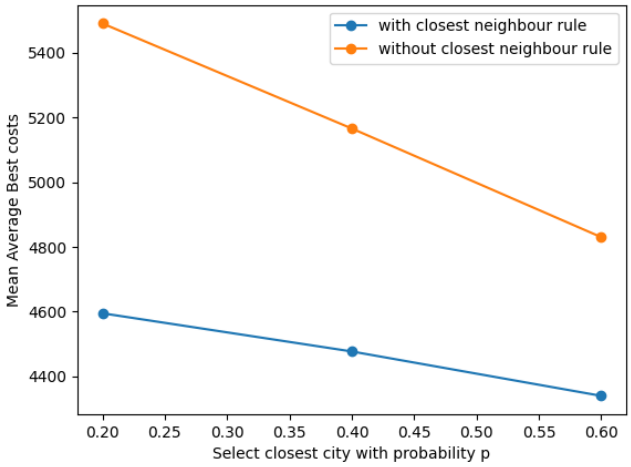

Figure 7: the influence q_0 on the mean average score

4.6.2 φ

φ is similar to ρ as it controls the decay of the pheromone of the local updates. As figure 8 shows a higher values, thus a longer endurance of the local pheromone is beneficial for the results.

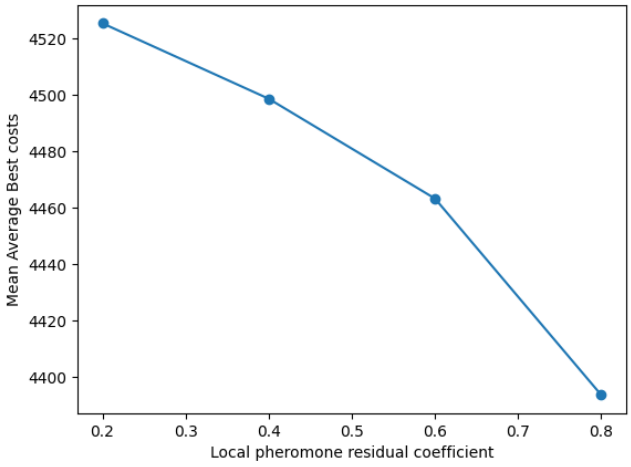

Figure 8: the influence of φ on the mean average score

5 Discussion

Our results give a good impression on how the different parameters regulate the algorithms' behaviour. However, if we compare our individual findings on the parameters with our best results we see a discrepancy in the used parameters. A possible reason is that we were not able to run the algorithms multiple times with the same parameters. Also we cannot derive direct causality from our findings.

We also observed a positive effect of the closest neighbours list on computation time as well as on the quality of the results. This is especially interesting, as it can be pre-computed from a distant matrix and thus will not increase the computation time. A major problem would be space.

Time and space is in general a problem with this kind of algorithms. This came to no surprise as it is an NP-hard problem.

Another interesting observation is the influence of α especially on the Min-Max algorithm. We assumed this parameter would be the most influential one among all parameters, as it controls how strong the pheromone affects the decision of the ants. In the case of the Min-Max algorithm that does not seem to be the case.

5.1 Conclusion and Future Work

In this project we implemented the main functionalities of the three ACO algorithms of our interest with minor difficulties, thanks to the rich and well documented background literature. Unfortunately, the same implementation would never work on the initially desired number of cities and thus, we were faced with long run-times and countless memory errors. Moreover, we introduced some approaches on the much smaller set of cities that had a positive influence on the results, especially the use of the Closest Neighbours.

However, all our findings are empirical which makes it hard to have any specific conclusion. Also the number of cities is way smaller than we had hoped to manage. The main reason for this is the limitations of equipment that was available to us. Another setback was the loss of one of our team members halfway through the project. We hoped to achieve more in this project but believe it does reflect our efforts.

To resolve the issues we had, the use of more optimisations could help. For example the paper [14] proposes some heuristic improvements. Such as using the pheromone information to improve the local search. The pheromones are used to compute a set of possible candidates. As the 2-opt local search is increasing the computation time and especially with a large number of nodes not very efficient our guess is that a reduction of possibilities will have a similar effect as the closest neighbours' computation.

Another idea would be to combine ACO with other optimisation algorithms. This is done by [15]. ACO is combined with genetic algorithm and Particle swarm optimisation. The main idea is to find promising candidates that are then improved by ACO.

An approach especially for a large amount of cities is the use of clustering methods to leverage properties of the data. [16] clusters dense parts of the large dataset into subsets that are then solved individually. A more conservative idea is running computations in parallel and use a more powerful GPU [17] [18]. Ant colony optimisations are well suited for parallelisation as in theory each ant is independent of the others and thus the local construction of the candidates can be done in parallel.

References

[1] G. Kizilates and F. Nuriyeva, "On the nearest neighbor algorithms for the traveling salesman problem," *Advances in Intelligent Systems and Computing*, vol. 225, pp. 111–118, 01 2013.

[2] M. L. Pilat and T. White, "Using genetic algorithms to optimize acs-tsp," in *Ant Algorithms* (M. Dorigo, G. Di Caro, and M. Sampels, eds.), (Berlin, Heidelberg), pp. 282–287, Springer Berlin Heidelberg, 2002.

[3] R. Hassin and A. Keinan, "Greedy heuristics with regret, with application to the cheapest insertion algorithm for the tsp," *Operations Research Letters*, vol. 36, no. 2, pp. 243 – 246, 2008.

[4] M. Dorigo and C. Blum, "Ant colony optimization theory: A survey," *Theoretical Computer Science*, vol. 344, no. 2, pp. 243 – 278, 2005.

[5] M. Dorigo and L. M. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53–66, 1997.

[6] T. Stützle and H. Hoos, "Max-min ant system," vol. 16, 11 1999.

[7] T. St and M. Dorigo, "Aco algorithms for the traveling salesman problem," 04 1999.

[8] "Mona lisa tsp challange." http://www.math.uwaterloo.ca/tsp/data/ml/monalisa.html. Accessed: 13.5.2020.

[9] C. S. Kaplan and R. Bosch, "Tsp art," in *Renaissance Banff: Mathematics, Music, Art, Culture* (R. Sarhangi and R. V. Moody, eds.), (Southwestern College, Winfield, Kansas), pp. 301–308, Bridges Conference, 2005. Available online at `http://archive.bridgesmathart.org/2005/bridges2005-301.html`.

[10] M. Dorigo, "Optimization, learning and natural algorithms," *PhD Thesis,Politecnico di Milano*, 1992.

[11] B. Bullnheimer, R. Hartl, and C. Strauss, "A new rank based version of the ant system - a computational study," *Central European Journal of Operations Research*, vol. 7, pp. 25–38, 01 1999.

[12] Y. Yan, H. suk Sohn, and G. Reyes, "A modified ant system to achieve better balance between intensification and diversification for the traveling salesman problem," *Applied Soft Computing*, vol. 60, pp. 256 – 267, 2017.

[13] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 26, no. 1, pp. 29–41, 1996.

[14] H. Ismkhan, "Effective heuristics for ant colony optimization to handle large-scale problems," *Swarm and Evolutionary Computation*, vol. 32, pp. 140 – 149, 2017.

[15] T. Saenphon, S. Phimoltares, and C. Lursinsap, "Combining new fast opposite gradient search with ant colony optimization for solving travelling salesman problem," *Engineering Applications of Artificial Intelligence*, vol. 35, pp. 324 – 334, 2014.

[16] C.-Y. Pang, B.-Q. Hu, J. Zhang, W. Hu, and Z.-C. Shan, "Applying data clustering feature to speed up ant colony optimization," *Abstract and Applied Analysis*, vol. 2014, pp. 1–8, 05 2014.

[17] H. Bai, D. OuYang, X. Li, L. He, and H. Yu, "Max-min ant system on gpu with cuda," in *2009 Fourth International Conference on Innovative Computing, Information and Control (ICICIC)*, pp. 801–804, 2009.

[18] Y.-S. You, "Parallel ant system for traveling salesman problem on gpus," 01 2009.

A Parameters

The following values were used to perform a grid search on 39 cities:

- ant count = 10
- generations = 100
- α = [0.2, 0.4, 0.6, 0.8, 1.0]
- β = [1.5, 2.0, 2.5]
- ρ = [0.2, 0.4, 0.6, 0.8]
- φ = [0.2, 0.4, 0.6, 0.8]
- q_0 = [0.2, 0.4, 0.6]
- min = [0.0, 0.1, 0.2]
- max = [0.6, 0.8, 0.9, 1.0]

	with 10%		without 10%	
algorithm	avg time (s)	avg best score	avg time (s)	avg best score
AS	2.76	3705.03	2.86	4014.53
ACS	48.13	3671.74	50.37	3947.33
Max-Min	3.7	3882.46	3.85	4849.72

Figure 9: Average time and average best score of the small 39 city instance