UKRAINIAN CATHOLIC UNIVERSITY

FINAL PROJECT

Ant colony optimization algorithms

Authors: Viktoria Kocherkevych, Yaroslav Klym

Mentor: Oleksii Ignatenko

Department of Computer Sciences
Faculty of Applied Sciences

Lviv 2023

Abstract

This is a project for an Architecture of Computer Systems course, and the main topic of this project is to introduce ant colony optimization (ACO) algorithms. The project report starts with an introduction, then we describe the first problem we solved using the Ant System (AS) while showing sufficient results of experiments. Then we introduce the second problem, discuss solutions and show results. The code of the project can be found here: Github

Contents

Chapter 1

Introduction

ACO algorithms are well-known in the field of optimization algorithms. They are often misunderstood as being evolutionary algorithms, which is not true. So how do they work?
Ants are pretty simple creatures, which can efficiently survive only while working in groups. We will emphasize their special way to navigate to food. Each ant leaves a pheromone while it moves. Then, other ants can sense the pheromone on the ground and choose the way, where there are more pheromones. Therefore, they will follow the way other ants followed more frequently and after some time they will construct the shortest way.[4]
Inspired by ants, the Ant Colony Optimization algorithms are frequently used when the shortest path is needed.

Chapter 2

Traveling salesman problem

2.1 Description

The first problem we consider is the well-known combinatorial optimization problem - The traveling salesman problem(TSP). As was mentioned before, the ACO algorithms are frequently used for finding the shortest path. This is the case. The traveling salesman problem can be formulated in the following way: there is a complete undirected weighted graph. The question is: find the Hamiltonian cycle with the minimum total distance.
Since finding the Hamiltonian cycle is classified as an NP-hard problem, the computation of an optimal solution for large graphs in a reasonable time is hard. Therefore, we will use the Ant colony optimization algorithms (which are stochastic) to find the close-to-optimal solution for the TSP.

2.2 Our solutions

2.2.1 Ant system

Ant system (AS) is the simplest first version of the algorithm.
During the set number of iterations(generations), we send "ants" to randomly go through the graph. The probability of going from node i to node j through edge e_{ij} is determined by the amount of pheromone on this edge and the distance between these nodes and calculated using the formula:

$$p_{ij} = \frac{\phi_{ij}^{\alpha} / d_{ij}^{\beta}}{\sum_{m=1, m \notin C_k}^{n} \phi_{im}^{\alpha} / d_{im}^{\beta}}$$

where ϕ_{ij} - the amount of the pheromone on the e_{ij}, d_{ij} - the distance between nodes i and j, C_k - set of nodes that already were added to the path, α - pheromone importance constant, β - heuristic information constant. Pheromone importance and heuristic information constants are used to determine the relative importance between pheromone amount and distance between nodes.
Then, after every ant constructs the path, we distribute pheromones between every node. If ant n went through e_{ij} the amount of the pheromone added to this edge is calculated by the formula:

$$\Delta\phi_{ij} = Q / L_n$$

, where $\Delta\phi_{ij}$ - amount of pheromone added, L_n - a total distance of the path created by ant n, Q - deposition constant, number of the pheromone added, for a total

distance 1. At the start of the simulation, edges already have some amount of the pheromone, which is determined by the initial pheromone constant. Also, after each generation, some amount of pheromone evaporates. Evaporation is implemented by the formula:

$$\phi_{ij} = (1 - \rho)\phi_{ij}$$

where ϕ_{ij} - amount of the pheromone on the e_{ij}, ρ - evaporation rate constant.

2.2.2 Mutations

Among modifications of AS implementation, first, we will discuss mutations. Every ant has some probability of being mutated. Then it will ignore pheromones and distance between nodes and go through the graph absolutely randomly. Mutations are implemented using a mutation rate constant, which is the probability of getting mutated. This ensures some amount of randomness and the possibility of finding a better solution on the other path.

2.2.3 Elitism

Elitism is another modification of AS. The idea is on each turn to choose some ants that found the shortest paths and then force them to repeat the same path in the next generation. This ensures that the best route will not be forgotten between generations of ants. The number of elite ants is also given with other constants used.

2.2.4 Ant colony system

Ant colony system(ACS) is a modification of AS that is based on two important extensions. First is that as ants traverse through the graph, they leave some amount of "negative" pheromone so that the next ant will go through this edge with a lower probability. That ensures that ants will not choose the same path every time. The amount of pheromone on the edge after the ant goes through it is calculated by the formula:

$$\phi_{ij} = (1 - \theta)\phi_{ij} + \theta\phi_0$$

where ϕ_{ij} - amount of the pheromone on the e_{ij}, ϕ_0 - initial pheromone, θ - pheromone decay constant.

Another extension is the introduction of the exploration constant. At every step of traversing the graph, there is some probability that the ant will choose the edge with the biggest amount of the pheromone with a 100 percent chance.

2.2.5 Max-Min ant system

The last modification of AS we will discuss is the Max-Min ant system(MMAS). This modification has two main features. First is that pheromones are distributed only between the edges of the lowest cost path. Second, the amount of the pheromone on the edge is bounded. There are two more constants: min pheromone and max pheromone, that don't allow the amount of pheromone on the edge to be lower than min and greater than max.

2.3 Algorithms comparison

The most important metric for comparison of the algorithms is the *length of the paths found*. In pictures 2.1-2.4, there are graphs that compare the minimum found path and most chosen path at each generation. The lower boundary on the y-axis is the minimum possible path. On the graph, you can also see the variance lines. As we can see minimum found paths for all algorithms are quite similar, which shows that all of the algorithms are working well.

2.4 Quality comparison with other solutions

In addition to ant colony algorithms, we implemented the simplest genetic algorithm to solve the TSP. It was implemented because we wanted to compare the performance of the ACO algorithm and the genetic algorithm.
The genetic algorithms are metaheuristic optimization algorithms that are inspired by evolution and nature. There are a few important things to remember about them and this algorithm specifically: the generation is initialized, fitness evaluation (in this case, it is better to follow the shortest path), then there is selection, mutation, new generation, and so on.[3] It was proved by us that the genetic algorithm, because of its nature, works faster but returns less accurate results. You can see on graph 2.5, that for the result close to ACO algorithms on the 10th generation, you need 300 generations here.

2.5 Efficiency comparison (Parallelization)

In this section, we will discuss the effectiveness of the parallelization method we chose.
In this project, we used boost::asio::threadpool[2]. The part of the algorithm which we made parallel is the traversing of the graph by ants. We put ants in the different threads. The only algorithm we didn't parallelize is ACS, as this algorithm needs communication during traversing when ants put "negative pheromones" on the edges. Picture 2.5-2.10 demonstrates time dependency to the number of threads and effectiveness of parallelization

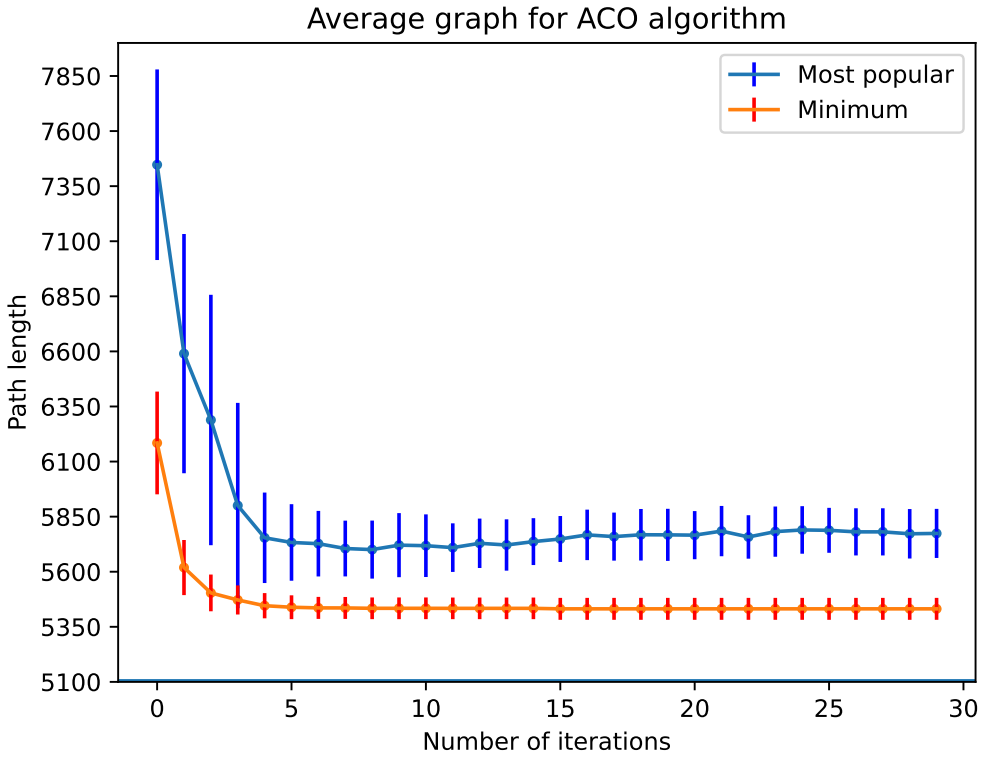

FIGURE 2.1: ACO

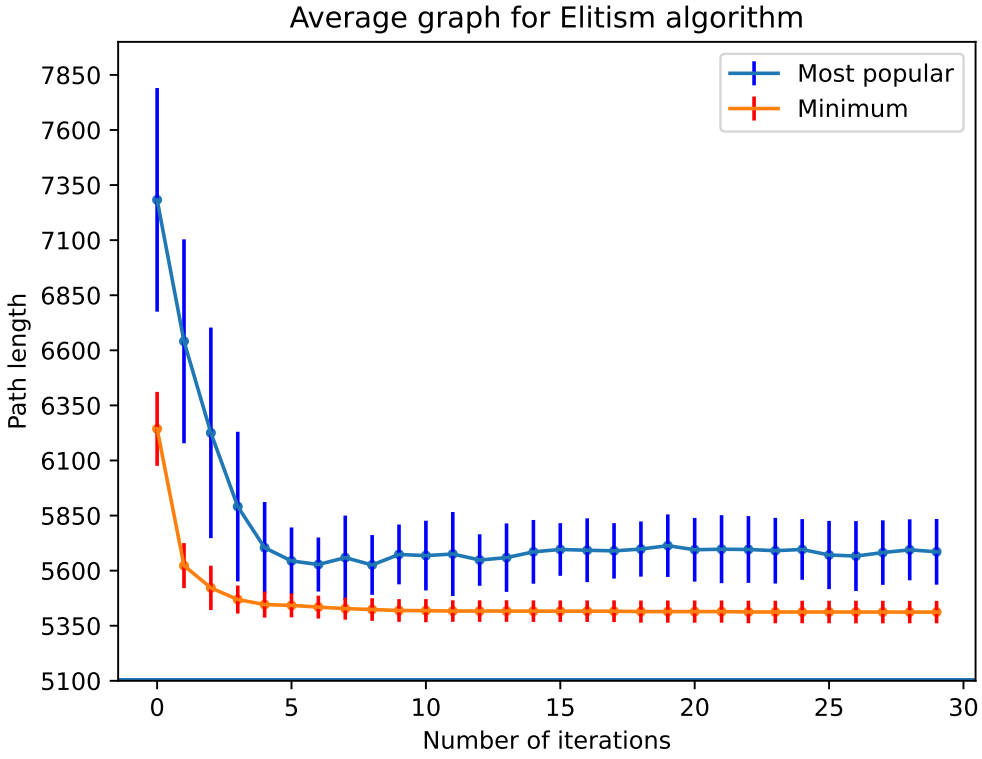

FIGURE 2.2: Elitism

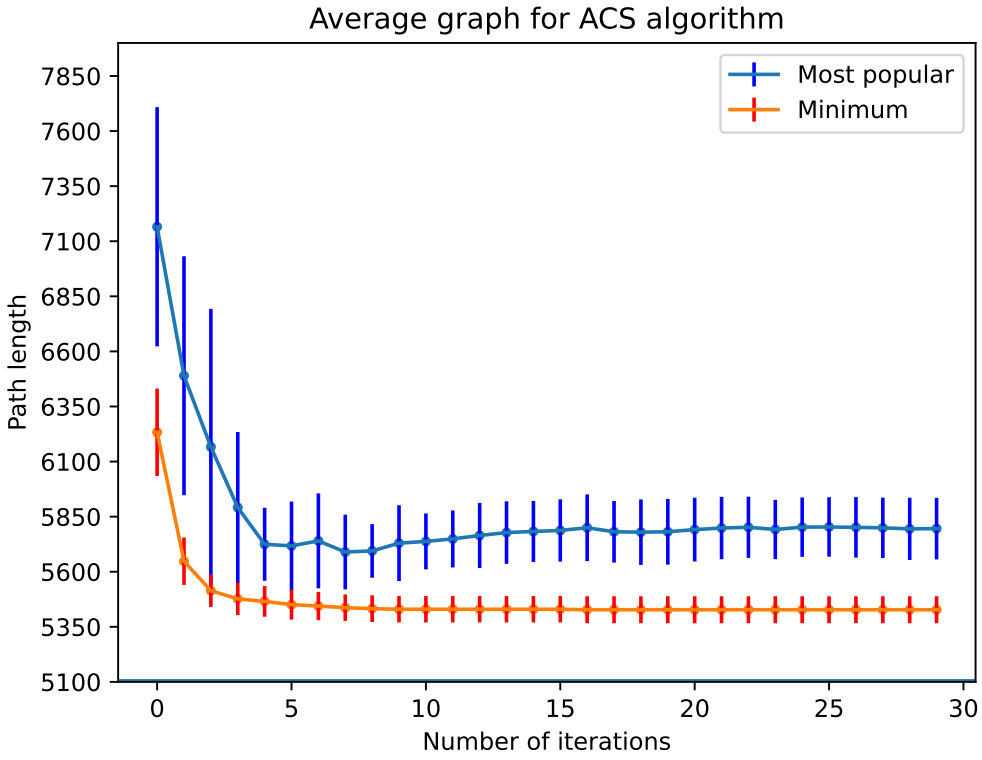

FIGURE 2.3: ACS

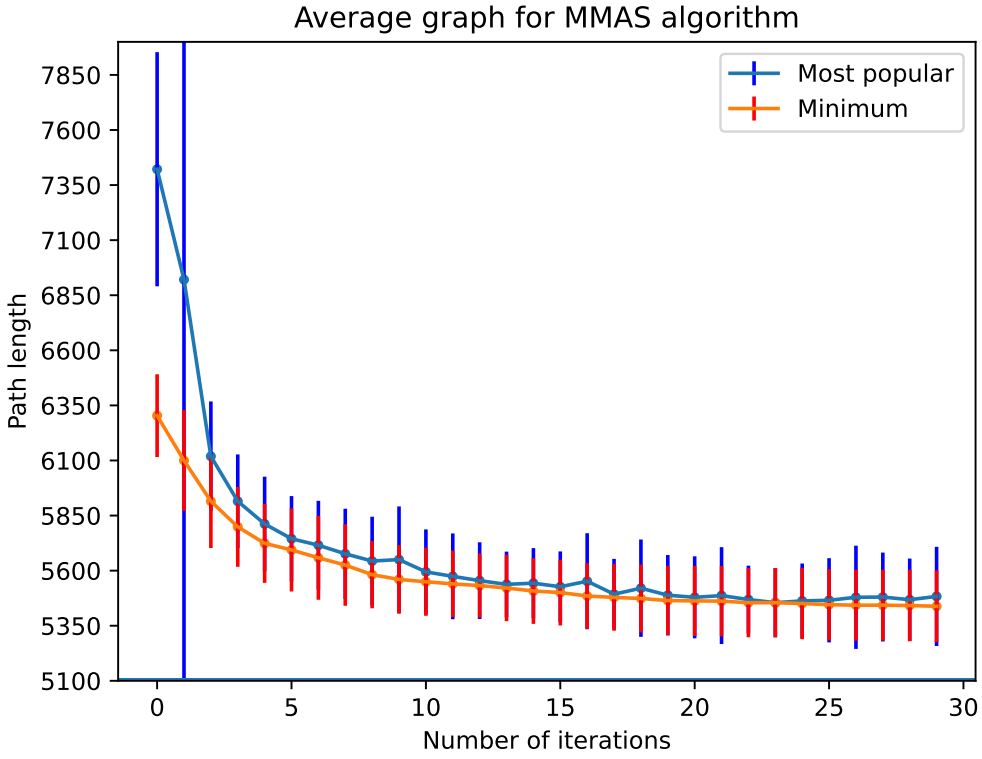

FIGURE 2.4: MMAS

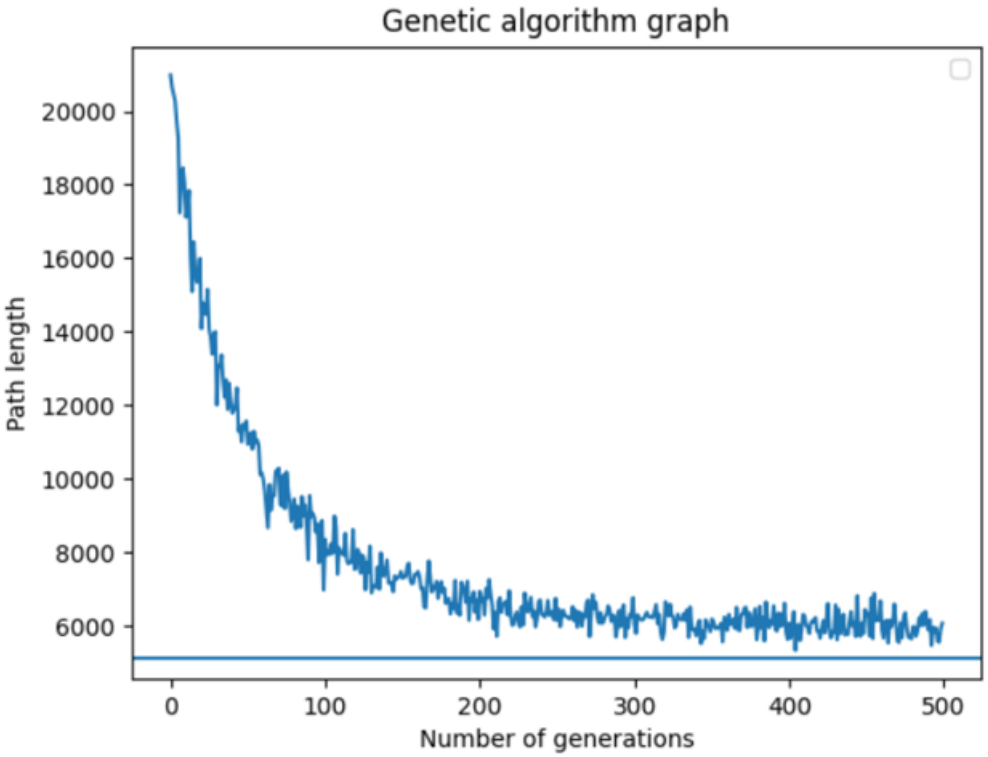

FIGURE 2.5: Generation to shortest path comparison

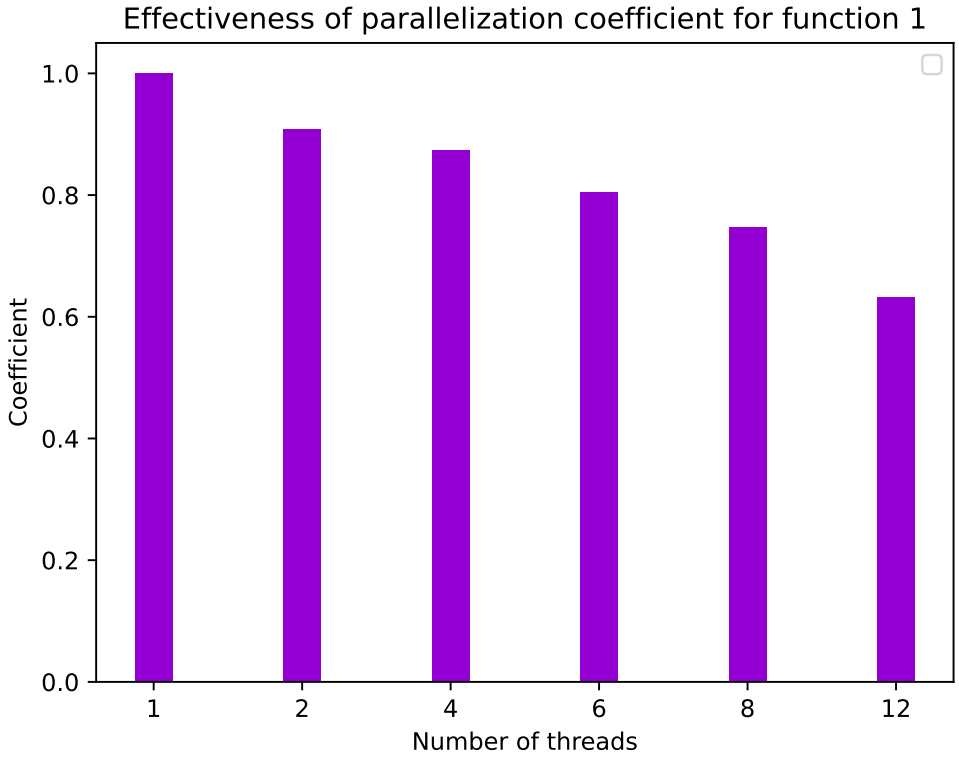

FIGURE 2.6: Efficiency for ACO

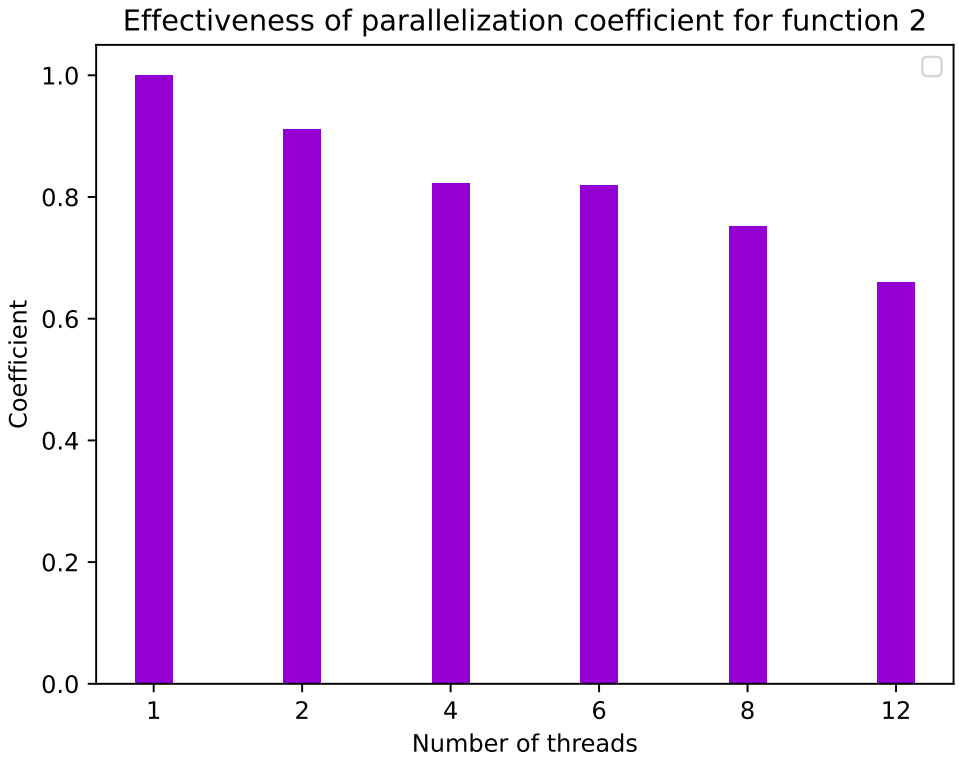

FIGURE 2.7: Efficiency for Elitism

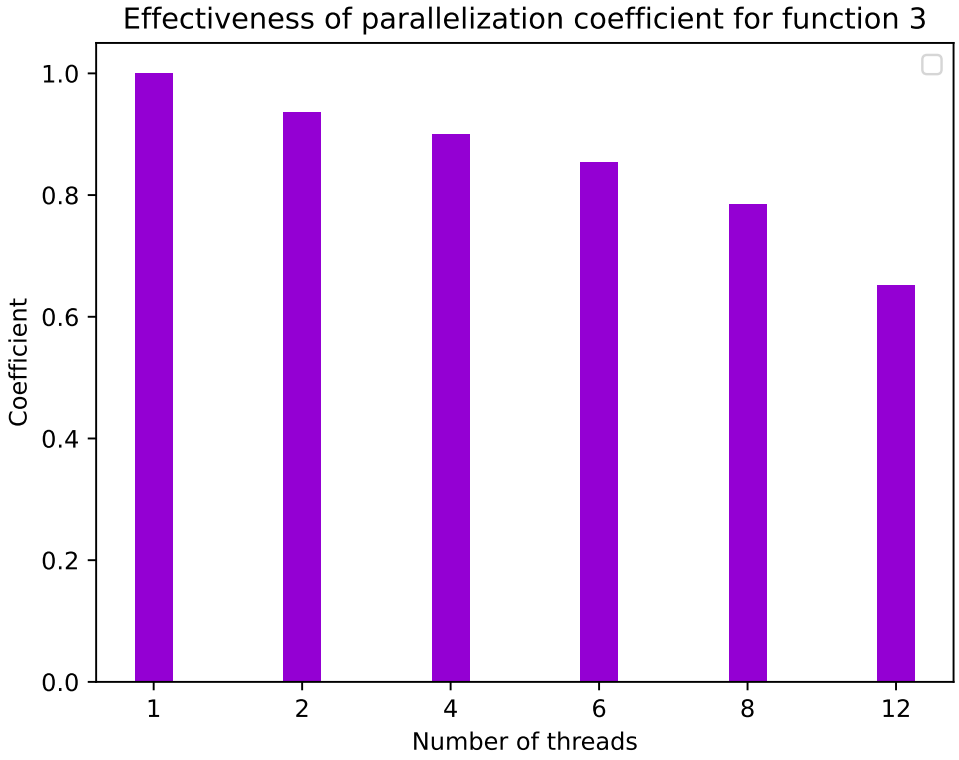

FIGURE 2.8: Efficiency for MMAS

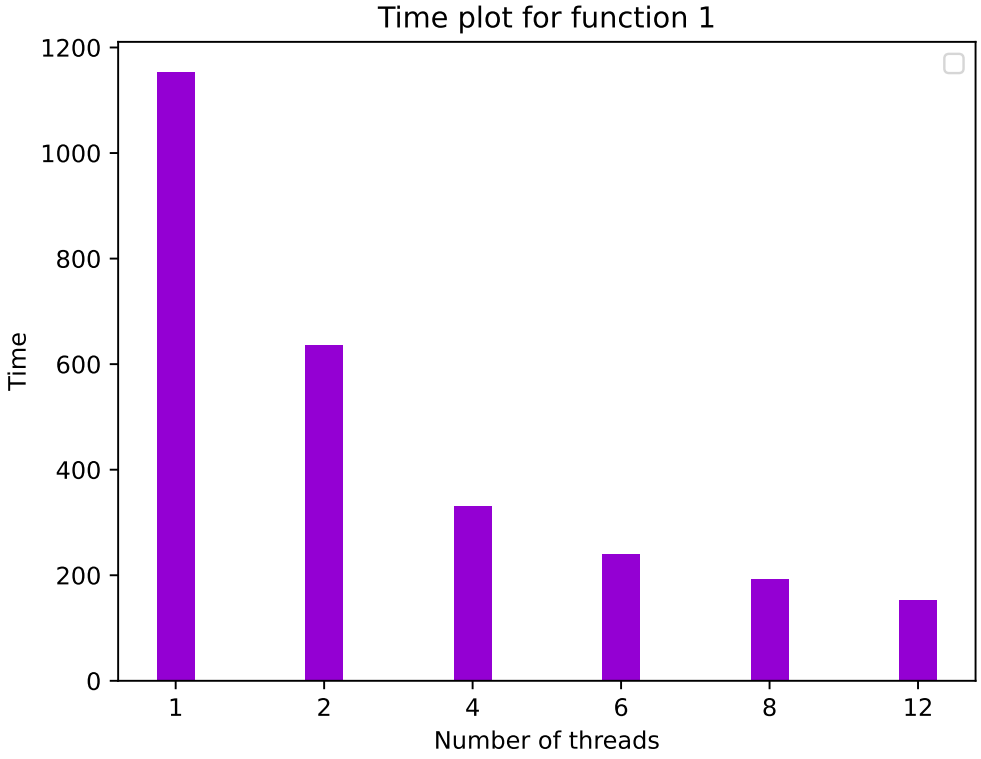

FIGURE 2.9: Time dependency of ACO

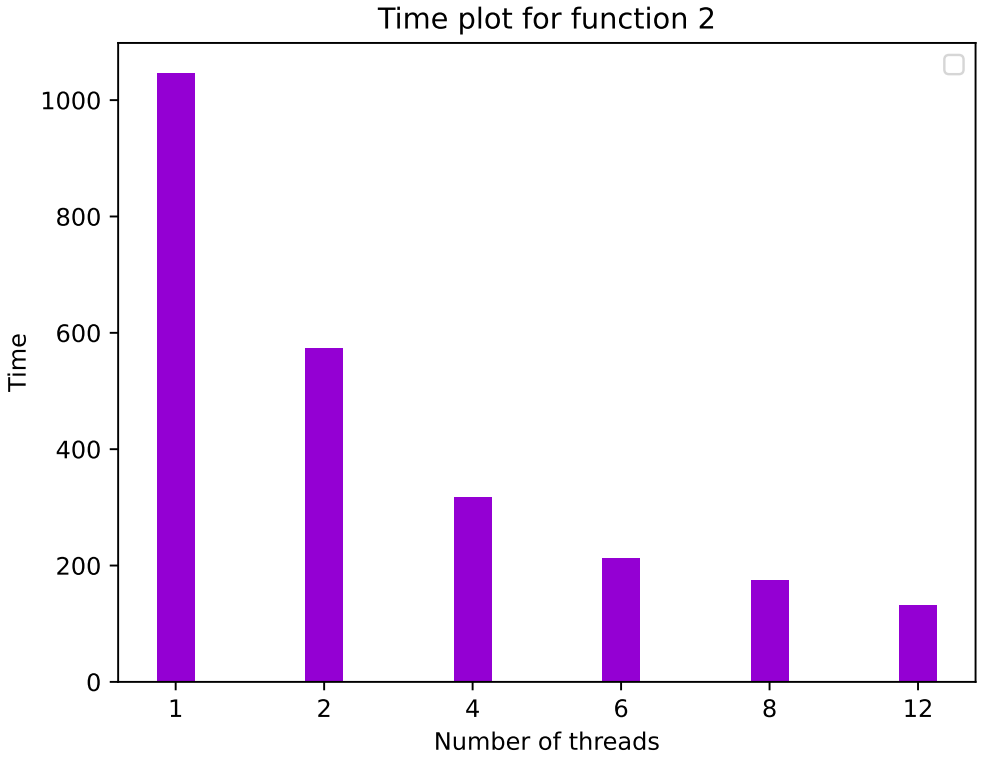

FIGURE 2.10: Time dependency of Elitism

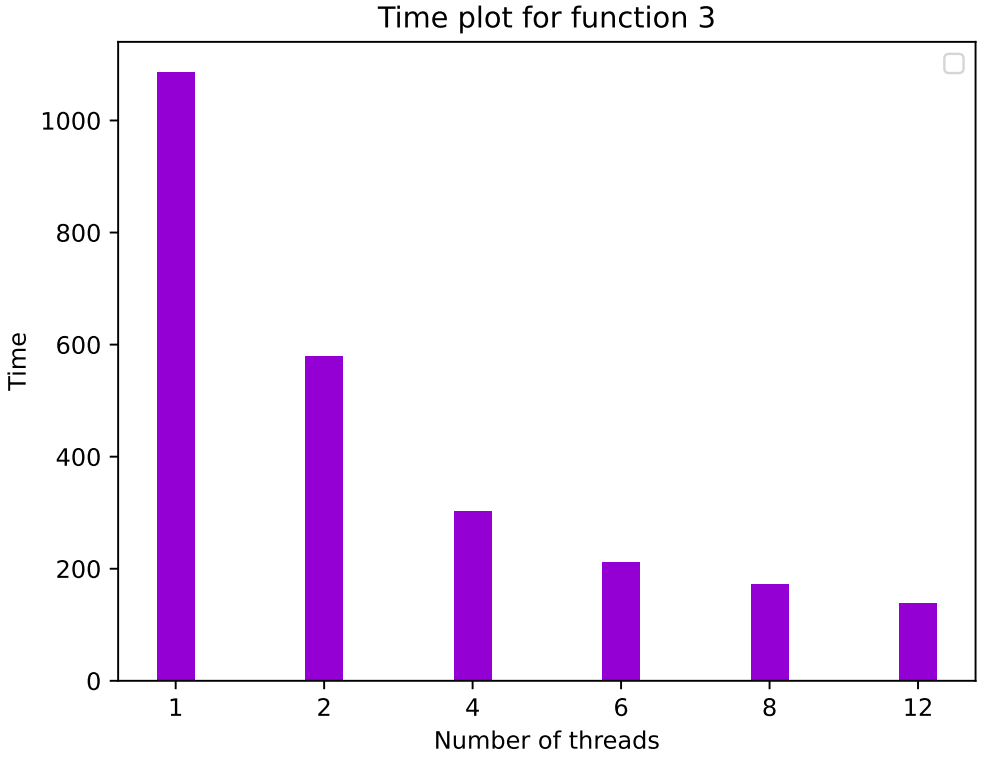

FIGURE 2.11: Time dependency of MMAS

Chapter 3

Stable matching problem

3.1 Description

The Stable matching problem (also called the Stable marriage problem) is a relevant problem, which tends to occur in real life. It can be described in terms of the following rather typical situation: n applicants apply to m universities, and the admission office has to decide which ones to admit to it.[1] Both universities and applicants have their preferences (some conditions based on which they can decide whether the other is worthy to be chosen. The problem is: how to make *stable* correspondences, where everybody will be satisfied with the result.
We consider slightly another formulation of the problem: a stable marriage problem, where there is a need to find the most profitable stable pairings of men and women. The stability condition is defined as follows:
*The pairing of men and women will be called unstable if there exists a woman w who prefers a man m*1 *over her current partner and that man m*1 *prefers the woman w over his current partner.* [1]
The authors of the cited article are also the authors of the main algorithm used to solve this problem: The Gale-Shapley algorithm, which finds a stable matching in time complexity of $O(n^2)$. We will try to solve this problem using ant colony optimization and provide some comparison with already existing solutions.

3.2 Our solution

3.2.1 Min preference

Our first solution is quite similar to AS for TSP. The first step is creating a bipartite graph where on the left side are men and on the right side are women. Edges show pair, and the value of the edge is a sum of the preferences of both members of the pair. Then when we have a graph, we send ants from all "man" nodes to "woman" nodes. As in the ant system for TSP, the probability that the ant will choose is determined using the value of the edge and the amount of the pheromone. After ants construct paths, pheromones are distributed between edges. This algorithm finds the minimum of the sum preference between all pairs but not a stable matching.

3.2.2 Offer and accept

Similar to the Gale-Shapley algorithm, this algorithm starts with sending offers to the opposite side, and then they can be accepted or declined. We also start with a bipartite graph, but the difference is that now it is directed. On one side, there is the

preference for men, and on the other, the preference for women. The first step is that ants from the men's nodes of the graph build paths to the opposite node. Then they distribute pheromones between nodes, and the turn goes to the women nodes. Now they build paths and one more time distribute pheromones. In the end, we one more time go through the graph choosing nodes with the biggest amount of pheromones.

3.3 Quality comparison with another solution

We implemented the original Gale-Shapley algorithm and performed some comparisons to our method.
First of all, the Ant colony optimization cannot return the full list of stable matchings. Stability is a quality, which cannot be fully satisfied using our implementations, therefore, we chose it to be one of the metrics of quality of an algorithm.

The second metric we used to build some conclusions about the work of the algorithms was the average preference number of the partner chosen. Where the lower the number, the most preferable partners were chosen.

We put both metrics on the graphics 3.1-3.3

We can see that the second algorithm gives the solution in terms of average preference more like Gale-Shapley, but give less stable result. Min preference actually works not that badly, but we can see that the bigger number of pairs - the less stable pairs.

3.4 Efficiency comparison

If we compare these two algorithms with Gale-Shapley in the way of the time they consume, we can say that actually, Gale-Shapley is still working faster. Algorithmically both algorithms $O(n^2)$. Also, we parallelized algorithms, but they were still slower than Gale-Shapley.

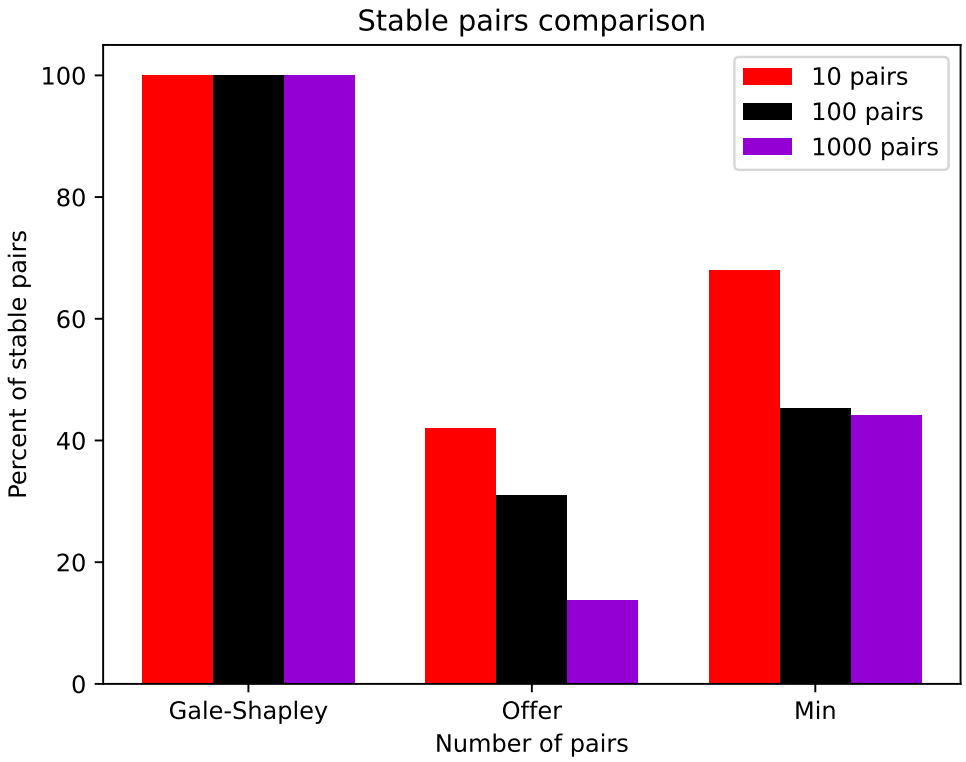

FIGURE 3.1: Percentage of stable pairs for different numbers of pairs

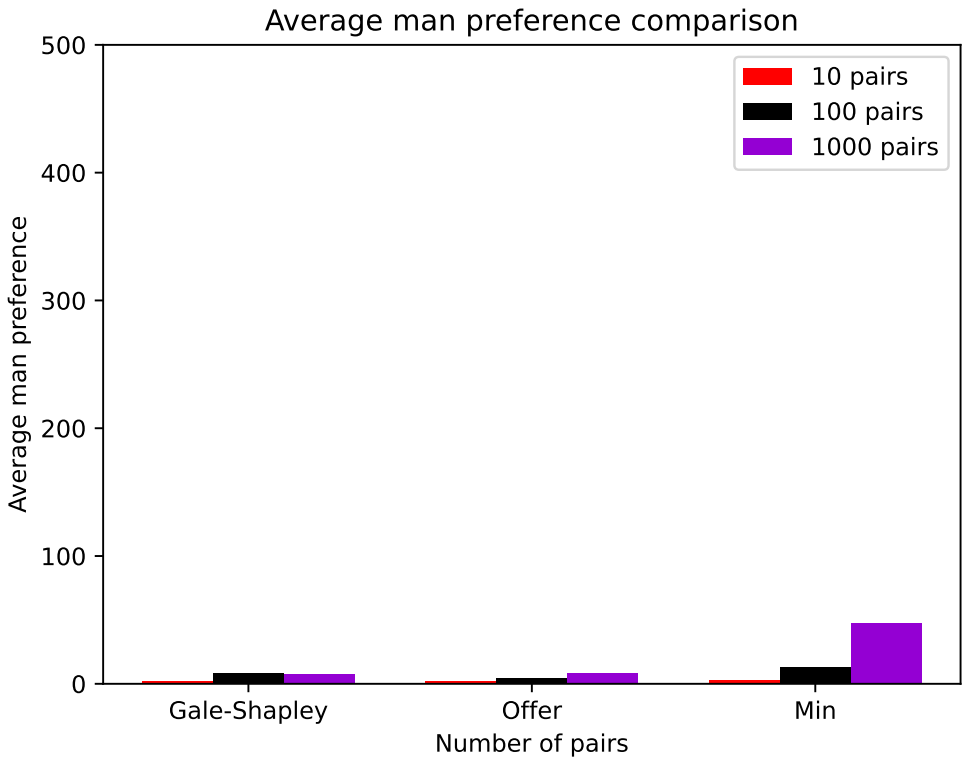

FIGURE 3.2: Average man preference on different numbers of pairs

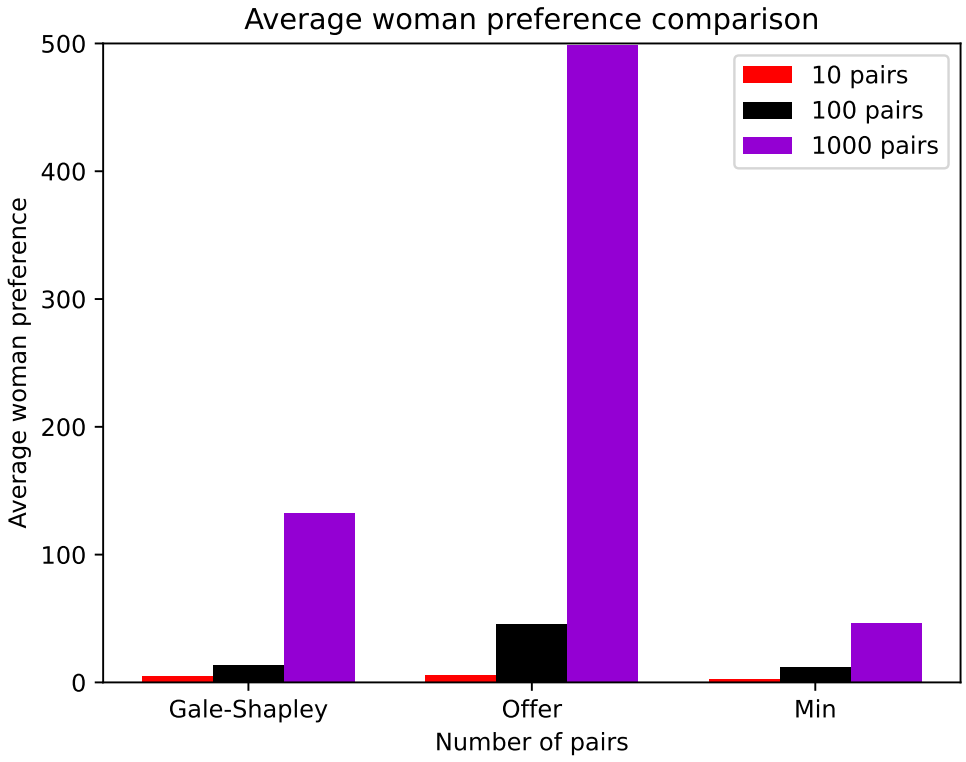

FIGURE 3.3: Average woman preference on different numbers of pairs

Chapter 4

Conclusions

In conclusion, this project explored the application of Ant Colony Optimization (ACO) algorithms to two classic optimization problems: the Traveling Salesman Problem (TSP) and the Stable Marriage Problem (SMP). The aim was to investigate if there is a reason to choose the ACO algorithm over other alternatives in solving those two problems.
To sum up, the first problem is classic and the ant colony solution is one of the common-known solutions to this problem. Here we proved once again that the ACO algorithms are suited to solve this problem (though they are still stochastic). Also, we tried to use ACO in some new for this algorithms field. But sadly, this algorithm didn't solve it well. It is slower and gives fewer correct solutions.

Bibliography

[1] D. Gale and L. S. Shapley. "College Admissions and the Stability of Marriage". In: *The American Mathematical Monthly* 69.1 (1962), pp. 9–15. ISSN: 00029890, 19300972. URL: http://www.jstor.org/stable/2312726 (visited on 06/01/2023).

[2] Christopher M. Kohlhoff. "threadpool". In: (2003-2017). URL: https://www.boost.org/doc/libs/1_66_0/doc/html/boost_asio/reference/thread_pool.html.

[3] Pedro Larranaga et al. "Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators". In: *Artificial Intelligence Review* 13 (Jan. 1999), pp. 129–170. DOI: 10.1023/A:1006529012972.

[4] Dan Simon. *Evolutionary Optimization Algorithms*. Wiley, 2013. URL: https://www.perlego.com/book/2759801/evolutionary-optimization-algorithms-pdf.