Pheromones, Mutations, and Tables: Exploration of TSP solution with use of ACO, GA, and DP

1st Marsel Berheev
Innopolis University
Innopolis, Russia
m.berheev@innopolis.university

2nd Makar Egorov
Innopolis University
Innopolis, Russia
m.eogrov@innopolis.university

3rd Nikita Stepankov
Innopolis University
Innopolis, Russia
n.stepankovv@innopolis.university

***Abstract*—The Traveling Salesman Problem (TSP) is one of the most studied NP-hard combinatorial problems, commonly used to benchmark optimization algorithms. In this paper, three different ways of solving TSP are examined: Ant Colony Optimization (ACO), Genetic Algorithm (GA), and Dynamic Programming (DP). These algorithms were compared based on several performance metrics such as execution time, memory usage, number of iterations, and solution correctness. The study aims to identify the most suitable algorithm for different conditions. Experimental results show that while DP provides exact solutions for small instances, ACO and GA offer scalable heuristics. Furthermore, ACO demonstrated great performance on large datasets compared to GA.**

***Index Terms*—TSP, ACO, GA, DP, NIC**

I. Introduction

The Traveling Salesman Problem (TSP) is a classic NP-hard problem in combinatorial optimization with both theoretical and practical importance [1]. Beyond benchmark purpure for evaluating algorithmic performance, TSP has real-world applications in fields such as transportation logistics, circuit design, and robotics.

The number of possible tours increases factorially with the number of cities, making brute force search performs badly except small problem sizes.

Due to its computational complexity, no polynomial-time algorithm is known to solve all instances of TSP optimally. However, several metaheuristic algorithms have been developed to find near-optimal solutions in a reasonable amount of time.

This study aims to evaluate and compare the performance of three different approaches to solving TSP: two nature-inspired algorithms and one exact method. The goal is to identify the strengths and weaknesses of each algorithm using various metrics, including execution time, memory consumption, number of iterations, and solution correctness.

A. Used Algorithms

This paper study two of the most widely used nature-inspired metaheuristic algorithms, together with a classical dynamic programming approach as exact solution:

- **Ant Colony Optimization (ACO)** – Originally introduced in the early 1990s for solving the TSP [2], ACO is inspired by the foraging behavior of ants. By simulating pheromone trail, the algorithm probabilistically constructs paths that able to approximate an optimal solution. While many enhanced versions of the original Ant System exist [3] [4], this study focuses on the basic implementation.
- **Genetic Algorithm (GA)** – Based on the principle of natural selection, GA evolves a population of candidate solutions over generations [5]. By applying operations such as selection, crossover, and mutation, the algorithm explores the search space to find near-optimal solution. Despite the availability of problem-specific enhancements for TSP [6] [7], this paper uses a standard GA without optimizations for fair comparison.
- **Dynamic Programming (DP)** – Used as a benchmark for solution quality, DP solves TSP exactly by exploring all possible permutations with the aid of memoization to avoid redundant computations. Although computationally expensive and slow for large problem instances, it guarantees the optimal tour and serves as a ground truth for evaluating the heuristic algorithms.

B. Modeling Scenario

This study models the optimization problem as a delivery task for autonomous robots in a real-world environment. Each robot must deliver packages to a set of locations (buildings), minimizing the total travel distance while make shure every node is visited.

II. Related Work

The Traveling Salesman Problem (TSP) has been extensively studied over the past decades, yet no polynomial-time algorithm is known to solve it. Among the most widely used approaches for tackling TSP are metaheuristic algorithms: Ant Colony Optimization (ACO) and Genetic Algorithms (GA).

ACO, introduced by Dorigo et al., has demonstrated strong performance on various combinatorial optimization problems, including TSP [2], [3], [8]. Genetic Algorithms, inspired by principles of natural evolution, have also been effectively used to solve TSP of varying complexity [9], [10].

Several comparative studies have studied the effectiveness of these algorithms on different dataset sizes. For example, Haroun et al. highlight the good performance of ACO on large-scale, while GA tends to perform better on small to medium-sized graphs [11].

In addition to standard applications, hybrid approaches that combine ACO and GA have also been explored to deal the strengths of both algorithms. These hybrid models aim to improve algorithm speed and solution quality [12], [13].

This paper builds on existing work by comparing ACO and GA with a Dynamic Programming (DP) baseline on a real-world dataset, aiming to evaluate their practical performance and scalability.

III. Experiments Methodology

A. Dataset

The dataset is based on the city of Innopolis. Specifically, coordinates of 36 buildings were manually collected using Yandex Maps. Node 0 represents the starting point for all algorithms — the Pyaterochka supermarket. The remaining nodes represent delivery destinations (buildings).

To compute the distances between nodes, the `geopy` Python library was used to generate a distance matrix based on the collected coordinates.

B. Code Structure

All algorithms were implemented from scratch in Python, using referencess from university laboratory materials. Python was chosen due to its flexibility, ease of prototyping, and access to useful standard libraries.

Two modules from Python's standard library were particularly helpful:

- `time` – Used to measure execution time of algorithms.
- `tracemalloc` – Used to track memory allocation of algorithm execution.

Each algorithm shares a common interface, allowing unified benchmarking and metric collection.

C. Performance Metrics

To evaluate the efficiency of each algorithm, the following metrics were used:

- **Execution Time** – Total runtime measured using the `time` module. A maximum execution time threshold was used to prevent large runtime for inefficient cases.
- **Memory Usage** – Peak memory consumption during execution, measured using the `tracemalloc` module.
- **Number of Iterations** – The number of iterations performed until stopping. This was recorded only for the Dynamic Programming (DP) approach, as both Ant Colony Optimization (ACO) and Genetic Algorithm (GA) used predefined iteration limits.
- **Correctness of Solution** – Quality of the solution. Since DP guarantees the optimal solution, ACO and GA results were compared against DP to assess approximation quality.

D. Pseudocode

The core benchmarking loop is unified across all algorithms and scales progressively from small graphs to the full dataset. The following pseudocode demonstrates the experimental pipeline. (see Algorithm 1)

Algorithm 1: Benchmarking Framework

```
Input: List of nodes N, MAX_EXECUTION_TIME
foreach algorithm in {ACO, GA, DP} do
    for s ← 3 to 36 do
        Generate graph with s nodes
        Start timer
        Run algorithm
        if execution time > MAX_EXECUTION_TIME
          then
            Terminate algorithm early
        Stop timer
        Record metrics
Output: Table of performance metrics
```

E. Experimental Environment

All experiments were conducted on a machine with an Intel Core i5-10400H CPU, 16 GB RAM, running Windows 10. Python 3.13.2 was used. Algorithms were executed sequentially to ensure fair resource allocation and consistent benchmarking.

F. Algorithm Parameters

The metaheuristic algorithms were configured with commonly used default values based on previous literature. For ACO, we used $\alpha = 1$, $\beta = 1$, evaporation rate $\rho = 0.5$, and 10 ants. GA was configured with a population size of 100, a mutation rate of 0.01, and tournament selection. Both ACO and GA were run for a maximum of 1000 iterations. Dynamic Programming used a memoization-based approach and was used as the ground truth for correctness.

More information could be found in project reposetory

IV. Analysis and Observations

A. Dynamic Programming

We begin by analyzing the performance metrics of the Dynamic Programming algorithm. Due to hardware constraints, we were only able to test graphs of up to 22 nodes. However, by fitting a polynomial model using `scikit-learn`, we predicted the expected time and memory requirements for solving a 36-node instance of the TSP.

As shown in Figure 1, DP demonstrates exponential growth in both execution time and memory usage with respect to the number of nodes. The estimated number of iterations for solving a 36-node TSP instance exceeds 7.8 billion, requiring approximately 14 hours of continuous computation and over 120 GB of RAM. This makes DP inefficient for larger problem sizes.

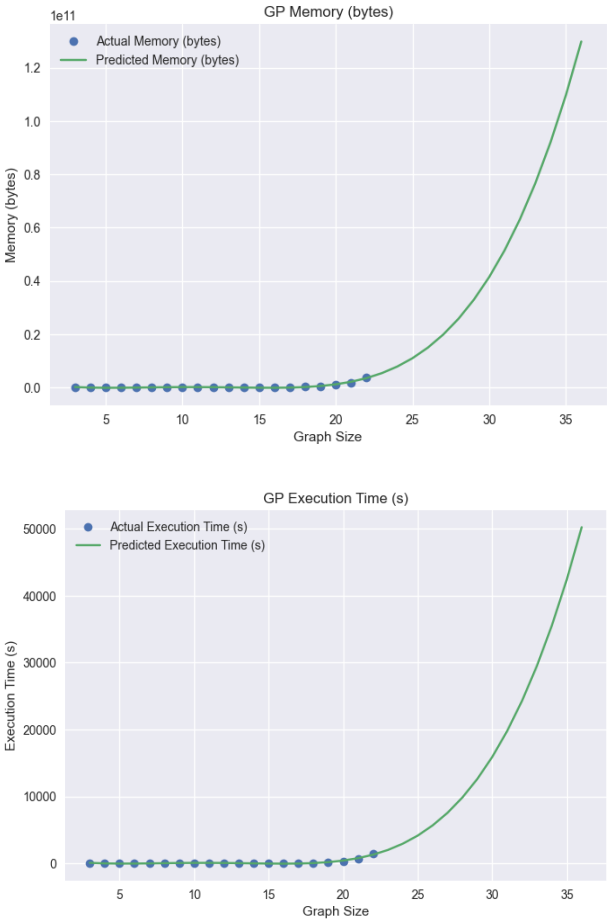


Fig. 1: Performance of the Dynamic Programming algorithm across different graph sizes.

B. Ant Colony Optimization and Genetic Algorithm

To evaluate the practical solution of heuristic methods, we compared the performance of Ant Colony Optimization (ACO) and Genetic Algorithm (GA) against the optimal solutions provided by DP.

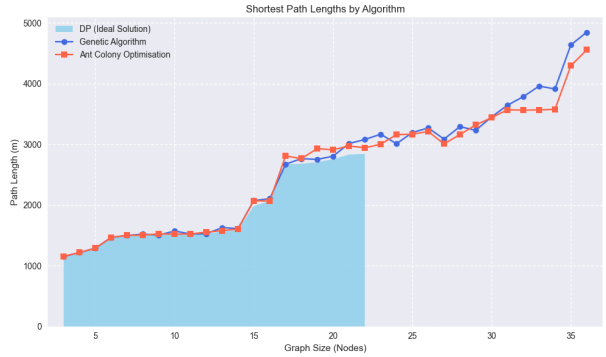
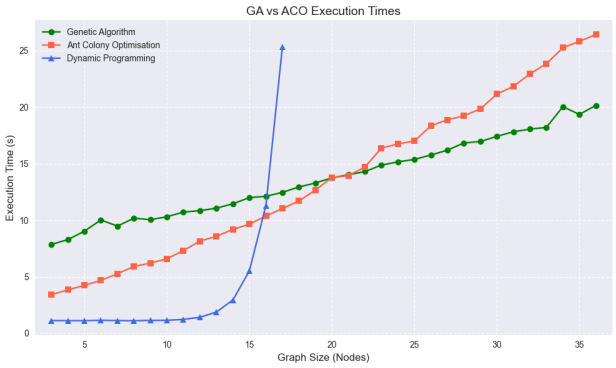


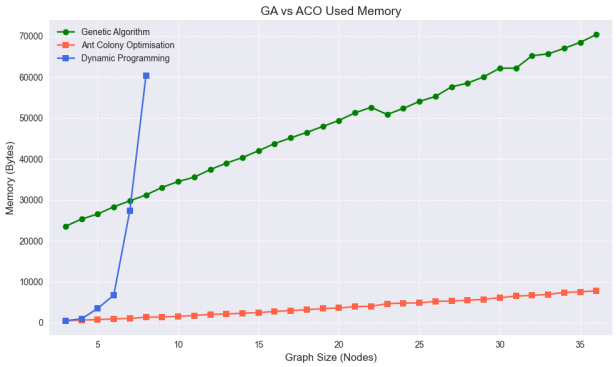
Fig. 2: Comparison of shortest path lengths found by Dynamic Programming, Ant Colony Optimization, and Genetic Algorithm.

As seen in Figure 2, both ACO and GA produce good solutions despite their stochastic nature and lack of global optimality. Their approximation quality remains close to the DP baseline, especially on smaller graph instances. This demonstrates the potential of metaheuristics for solving larger TSPs when exact methods become computationally expencive.

Now that we have established that both metaheuristic algorithms can provide high-quality solutions, we now take a closer look at their performance in terms of execution time and memory consumption.



(a) Execution time of ACO and GA



(b) Memory usage of ACO and GA

Fig. 3: Performance metrics of Ant Colony Optimization and Genetic Algorithm across varying graph sizes.

As shown in Figure 3, the GA demonstrates better performance on medium-sized graphs (up to 20 nodes), ACO outperforms it on larger graphs (above 20 nodes). It is also important to highlight that (DP) is most effective only for small graphs (up to 15 nodes), beyond which its resource requirements become too expensive.

Furthermore, nature-inspired algorithms outperform DP in terms of memory usage. As illustrated in Figure 4b, DP requires much more memory than either of the metaheuristic approaches. Between ACO and GA, ACO is more memory-efficient. This difference can be explained by the nature of the algorithms: GA is population-based and stores multiple solutions, while ACO maintains only a distance matrix and a pheromone matrix.

V. Conclusion

DP provided exact solutions for smaller graphs but became impractical for larger ones due to exponential growth in time and memory. GA showed stable memory usage and reasonable computation times, though its solution quality declined with larger graphs. ACO had similar performance to GA but generally consumed less memory and offered slightly better accuracy for mid-sized graphs.

While DP ensures optimal solutions, it lacks scalability. GA and ACO strike a balance between accuracy and performance, making them more suitable for large-scale problems where exact solutions are impractical. Depending on the specific requirements—accuracy, memory, or time—either GA or ACO may be more appropriate for large graphs.

TABLE I: Comparison of Algorithm Performance Across Graph Sizes 3 - 22

Graph Size	Dynamic Programming				Genetic Algorithm				Ant Colony Optimization			
	Iter.	Time (s)	Mem (MB)	Error	Iter.	Time (s)	Mem (MB)	Error	Iter.	Time (s)	Mem (MB)	Error
3	5	1.130	0.0005	0.0	1000	7.861	0.0225	0.0	1000	3.420	0.0005	0.0
4	16	1.120	0.0009	0.0	1000	8.284	0.0242	0.0	1000	3.848	0.0006	0.0
5	53	1.119	0.0033	0.0	1000	9.030	0.0253	0.0	1000	4.238	0.0007	0.0
6	166	1.151	0.0064	0.0	1000	10.025	0.0270	0.0	1000	4.678	0.0009	0.0
7	487	1.127	0.0261	0.0	1000	9.477	0.0284	0.0	1000	5.266	0.0010	0.0
8	1352	1.112	0.0575	0.0	1000	10.178	0.0298	14.27	1000	5.910	0.0013	0.0
9	3593	1.139	0.1521	0.0	1000	10.044	0.0315	0.0	1000	6.211	0.0014	18.88
10	9226	1.153	0.3591	0.0	1000	10.307	0.0329	50.28	1000	6.587	0.0015	0.0
11	23051	1.231	0.8004	0.0	1000	10.716	0.0339	0.0	1000	7.319	0.0017	0.0
12	56332	1.417	2.5479	0.0	1000	10.845	0.0357	0.0	1000	8.148	0.0019	27.08
13	135181	1.882	5.3703	0.0	1000	11.064	0.0372	65.06	1000	8.578	0.0020	14.79
14	319502	2.945	10.896	0.0	1000	11.449	0.0385	20.52	1000	9.186	0.0022	14.79
15	745487	5.530	22.216	0.0	1000	12.000	0.0401	87.60	1000	9.647	0.0024	84.92
16	1720336	11.293	46.237	0.0	1000	12.109	0.0417	38.41	1000	10.367	0.0026	0.0
17	3932177	25.327	95.968	0.0	1000	12.463	0.0431	0.0	1000	11.038	0.0028	133.91
18	8912914	55.486	199.008	0.0	1000	12.935	0.0443	85.19	1000	11.693	0.0030	86.51
19	20054035	126.525	411.773	0.0	1000	13.291	0.0457	46.18	1000	12.669	0.0033	223.93
20	44826644	284.085	852.396	0.0	1000	13.745	0.0471	50.28	1000	13.763	0.0034	155.77
21	99614741	636.632	1760.87	0.0	1000	14.020	0.0489	181.93	1000	13.943	0.0038	142.57
22	220200982	1407.401	3632.42	0.0	1000	14.298	0.0502	236.93	1000	14.700	0.0039	100.28

References

[1] A. P. Punnen, "The traveling salesman problem: Applications, formulations and variations," in *The traveling salesman problem and its variations*. Springer, 2007, pp. 1–28.

[2] 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.

[3] D. M. Chitty, "Applying aco to large scale tsp instances," in *Advances in Computational Intelligence Systems: Contributions Presented at the 17th UK Workshop on Computational Intelligence, September 6-8, 2017, Cardiff, UK*. Springer, 2018, pp. 104–118.

[4] R. W. Dewantoro, P. Sihombing *et al.*, "The combination of ant colony optimization (aco) and tabu search (ts) algorithm to solve the traveling salesman problem (tsp)," in *2019 3rd International Conference on Electrical, Telecommunication and Computer Engineering (ELTICOM)*. IEEE, 2019, pp. 160–164.

[5] J. H. Holland, "Genetic algorithms," *Scientific american*, vol. 267, no. 1, pp. 66–73, 1992.

[6] Y. Deng, Y. Liu, and D. Zhou, "An improved genetic algorithm with initial population strategy for symmetric tsp," *Mathematical problems in engineering*, vol. 2015, no. 1, p. 212794, 2015.

[7] Y. Yu, Y. Chen, and T. Li, "A new design of genetic algorithm for solving tsp," in *2011 Fourth International Joint Conference on Computational Sciences and Optimization*. IEEE, 2011, pp. 309–313.

[8] T. Stützle, M. Dorigo *et al.*, "Aco algorithms for the traveling salesman problem," *Evolutionary algorithms in engineering and computer science*, vol. 4, pp. 163–183, 1999.

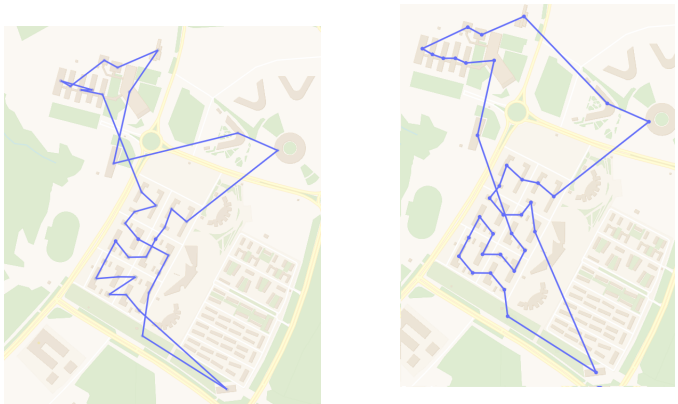
[9] N. M. Razali, J. Geraghty *et al.*, "Genetic algorithm performance with different selection strategies in solving tsp," in *Proceedings of the world congress on engineering*, vol. 2, no. 1. International Association of Engineers Hong Kong, China, 2011, pp. 1–6.

[10] U. Hacizade and I. Kaya, "Ga based traveling salesman problem solution and its application to transport routes optimization," *IFAC-PapersOnLine*, vol. 51, no. 30, pp. 620–625, 2018.

[11] S. A. Haroun, B. Jamal *et al.*, "A performance comparison of ga and aco applied to tsp," *International Journal of Computer Applications*, vol. 117, no. 20, 2015.

[12] D. Gong and X. Ruan, "A hybrid approach of ga and aco for tsp," in *Fifth World Congress on Intelligent Control and Automation (IEEE Cat. No. 04EX788)*, vol. 3. IEEE, 2004, pp. 2068–2072.

[13] M. Reimann, S. Shtovba, and E. Nepomuceno, "A hybrid aco-ga approach to solve vehicle routing problems," *Student Papers of the Complex Systems Summer School, Santa Fe Institute, Budapest*, 2001.



(a) GA Shortest Path (b) ACO Shortest Path

Fig. 4: Shortest paths generated by ACO and GA