Selforganizing Systems - Exercise 1

Felix Schmidt 12002283
Joachim Biberger 12347679

30. November 2025

1 Self-organizing Techniques

We chose the following self-organizing techniques to be used in this exercise:

Genetic Algorithm (GA):
Genetic Algorithms are optimization techniques inspired by the process of natural selection. They work by generating a population of potential solutions and evolving them over time using operations such as selection, mating, crossover, and mutation. The goal is to iteratively improve the solutions and converge toward an optimal result.

Ant Colony Optimization (ACO):
Ant Colony Optimization is a swarm intelligence–based algorithm inspired by the behavior of ants searching for food. These "Ants" explore possible solutions and communicate using pheromone trails. Over time, shorter or better paths receive stronger pheromone reinforcement, which guides the algorithm toward optimal solutions. It is particularly useful for discrete combinatorial optimization problems such as path finding or routing.

Particle Swarm Optimization (PSO):
Particle Swarm Optimization is another swarm-based method inspired by the collective movement of birds. Each particle represents a potential solution and moves through the search space influenced by its own best-known position and the best position found by the group. This cooperative behavior allows PSO to efficiently search for optimal solutions, especially in continuous search spaces.

1.1 Problem Tasks

We focused on the following problems as project tasks for our assignment:

Travelling Salesman Problem:
The Optimization-goal of the Travelling Salesman Problem is finding the shortest route that visits each city exactly once and returns to the starting point. This represents a complex combinatorial optimization challenge. As the number of cities increases, the search space grows exponentially, making it a problem that is computationally intense problem to solve. This allows self-organizing techniques such as GA, ACO, and PSO to demonstrate their ability to search efficiently and find near–optimal solutions in a large solution space.

TSP Dataset Description:
For the Travelling Salesman Problem, we used artificially generated city coordinates consisting of two-dimensional (x, y) points obtained from *[1]*. The dataset includes four separate instances of increasing size to evaluate how the algorithms scale with problem complexity. Each instance contains a different number of cities, ranging from very small to large problems. Table 1 provides an overview of the datasets used in our experiments.

Dataset Name	**Number of Cities**
Tiny	10
Small	30
Medium	100
Large	1000

Table 1: Overview of the artificially generated TSP datasets used in our experiments.

Rastrigin Function:
The Rastrigin function is a well-known mathematical test function used to evaluate the performance of optimization algorithms. This function contains many local minima, making it difficult to optimize and suitable to assess the ability of algorithms to escape local minima and converge to the optimal solution. The goal is to find the global minimum, located at the origin.

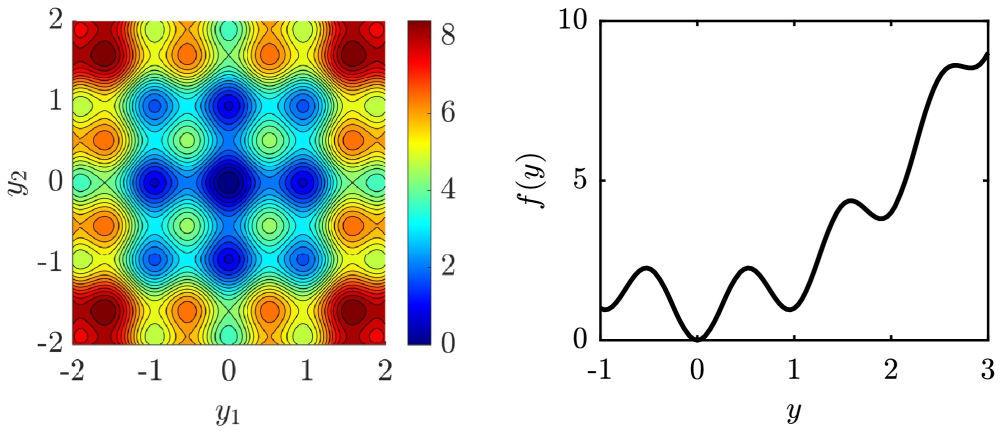


Figure 1: Illustration of the Rastrigin Function [2]

These two problems represent different types of optimization challenges: discrete combinatorial search and continuous multimodal minimization. This is why the combination of these allows us to assess how well the algorithms adapt to unique search spaces, escape local optima, and handle different problem structures. We used them to evaluate and compare the performance of all three self-organizing techniques.

2 Evaluation of techniques for TSP

Across all dataset sizes, the algorithms were able to identify valid tours, but the quality of the solutions differs noticeably, especially for the larger datasets. The identified best routes and convergence plots are included in the appendix.

2.1 Hyperparameters

The hyperparameters for the TSP experiments were selected based on a combination of recommended defaults from the *scikit-opt* library and empirical adjustments made during preliminary testing.
For the Genetic Algorithm (GA), a population size of 100 and 2000 iterations were used for the Tiny, Small, and Medium datasets to maintain sufficient diversity and ensure convergence within a reasonable runtime. The mutation probability of 0.1 follows standard GA practice and prevents premature stagnation. For the Large dataset, we increased the population size to 300 and the maximum number of iterations to 4000. This was done deliberately to bring the GA runtime closer to that of ACO and PSO, enabling a more comparable evaluation of solution quality at a similar computational budget.

For Ant Colony Optimization (ACO), a colony size of 50 ants and 150 iterations proved sufficient across all datasets, as pheromone reinforcement typically accelerates convergence on combinatorial problems. The distance matrix was provided directly to guide probabilistic path construction.

For Particle Swarm Optimization (PSO), larger settings were required because PSO is not inherently designed for permutation-based optimization. A population size of 200 and 800 iterations, combined with a high inertia weight ($w = 0.8$) and relatively small cognitive and social factors ($c_1 = c_2 = 0.1$), were selected to stabilize the discrete particle updates within the TSP-specific variant of PSO provided by *scikit-opt.* Overall, the chosen hyperparameters provide a reasonable trade-off between computational cost and solution quality and lead to consistent convergence behaviour across the TSP datasets.

2.2 Quality of Results

For the Tiny dataset (10 cities), all algorithms achieve essentially identical best distances (≈ 12.48), indicating that the search space is sufficiently small for all methods to converge reliably to the global optimum. As the dataset size increases, however, the algorithms diverge in performance.

Small and Medium datasets (30 and 100 cities): GA and ACO show comparable performance, with best distances in the range of 58.58 (GA) to 59.80 (ACO) on the Small dataset and around 8.11–8.48 on the Medium dataset. PSO, in contrast, consistently reaches worse solutions than GA and ACO, especially on the Medium dataset, where it obtains a noticeably higher tour length of 10.18. This reflects PSO's limitations when adapted to permutation-based optimization problems, as its continuous search dynamics make it more prone to premature convergence or unstable exploration.

Large dataset (1000 cities): The differences become most pronounced. ACO achieves by far the best solution with a tour length of 31.30, clearly outperforming both GA and PSO. With the more intensive hyperparameter setting, GA now finds a substantially better tour (75.95) than in the initial configuration (156.15), but still remains notably worse than ACO. PSO again performs the worst with a tour length of 219.21, which underlines its weak suitability for large-scale combinatorial problems in this setup.

Overall, ACO demonstrates the highest effectiveness on the large-scale TSP, while GA shows stable and reasonably strong performance across all sizes and can be improved further at the cost of runtime. PSO remains the least suitable method for TSP in this configuration.

Algorithm	Tiny	Small	Medium	Large
GA_TSP	12.4835667	58.5817759	8.4840642	75.9540395
ACO_TSP	12.4835670	59.7986190	8.1126870	31.2956640
PSO_TSP	12.4835667	58.0770380	10.1831664	219.2106570

Table 2: Best distances found by GA, ACO, and PSO for all TSP datasets.

2.3 Runtime Analysis

The runtime results reveal clear trends that reflect the computational design of each technique.

For the Tiny, Small, and Medium datasets, the fastest method is **ACO** (e.g., 0.93 s for Tiny, 3.46 s for Small, 16.29 s for Medium). GA follows with moderate runtime (between 2.75 s and 19.72 s), while PSO is always the slowest method in this range, with runtimes between 4.25 s and 23.40 s. The higher runtime of PSO is expected, since the PSO-TSP implementation requires additional repair, sorting, and feasibility corrections in each update step.

On the **Large dataset**, we tuned the GA hyperparameters to increase its runtime and thus make its computational effort more comparable to ACO and PSO. With the new settings, GA now requires 1353.30 s, compared to 819.54 s for ACO and 1007.49 s for PSO. In this configuration, GA becomes the slowest method, but all three algorithms now operate in a similar runtime regime (on the order of 10^3 s). This makes the comparison on the Large dataset more about solution quality at roughly comparable computational budgets, rather than about pure speed.

Algorithm	Tiny	Small	Medium	Large
GA_TSP	2.749729	6.657442	19.718001	1353.304460
ACO_TSP	0.932431	3.455379	16.292521	819.542114
PSO_TSP	4.250235	7.453415	23.395603	1007.489014

Table 3: Runtime of GA, ACO, and PSO for all TSP datasets (in seconds).

2.4 Conclusion

The evaluation shows that each self-organizing technique exhibits distinct strengths and limitations for the TSP:

- **ACO** achieves the best overall solution quality, especially on the Large dataset, and is very efficient on smaller instances.
- **GA** behaves robustly across all dataset sizes With moderate hyperparameters it offers a good balance between speed and accuracy, while still being outperformed by ACO in each case.
- **PSO** is the least suitable method for TSP in this setup. It consistently produces weaker solutions and exhibits relatively high runtime due to its less natural adaptation to permutation spaces.

In conclusion, for combinatorial optimization tasks such as the TSP, ACO is the strongest choice for best solution quality and performance. GA performs well, but is still outperformed for each dataset by ACO. PSO, while effective in continuous domains, is not competitive for TSP compared to GA and ACO in the tested configurations.

3 Evaluation of techniques for Rastrigin

For our implementation of the GA and PSO we adapted parts of the implementation from [3]. However, in order to run the ACO-algorithm on the rastrigin function, we did not find a preexisting GitHub repository that matched our implementation-style. This is why we used the implementation from [4] and adjusted it to fit into our implementation. We used this work-around since the goal of this exercise was focused more on comparing the different methods, instead of implementing them from scratch.

We ran each optimization-algorithm for 400 iterations, with the assumption that after these number of iterations the algorithm either found the best solution or converged to a local minima. To verify this assumption, we plotted the convergence of the best found-minima for each iteration. These can be seen in the figure below:

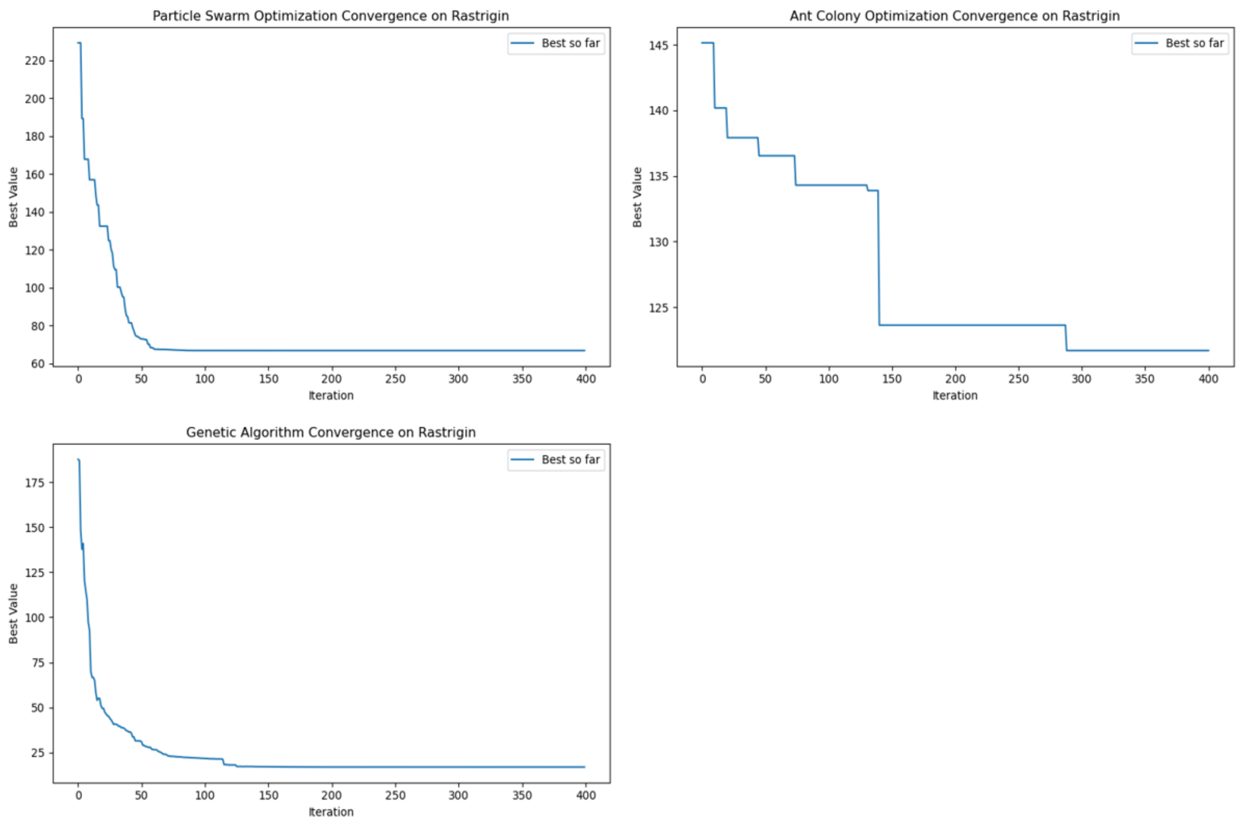


Figure 2: Convergence towards the Minima of GA, ACO and PSO on the 15-dim Rastrigin Function

In Figure X we can clearly see a convergence towards the final Minima in the form of an exponential decay for all self-organizing techniques. However, looking at the scale we can see that some techniques arrive at their Minima more efficiently than others. Also the quality of the reuslts differ. What is consistent across all three techniques is that changes per iteration rapidly decrease and after around 150 iterations they are very small, which makes it reasonable to stop the algorithms at that point or after (e. g. stop at 400 iterations).

In order to facilitate a fair comparisson we set the population-size for each algorithm to 100. This is a fairly large size which is needed to perform well on tasks with high dimensionality and many local minima like the rastrigin function. The other parameters were evaluated by reading into literature and lecture notes and testing the performance of the algorithms with different parameter-settings for various dimensionalities of the rastrigin function. We then decided on the parameters shown in Table X with which reasonable good and stable solutions were achieved for the different dimensionalities tested.

Model	Parameter	Value
GA	size_pop	100
	max_iter	400
	prob_mut	0.77
ACO	n_ants	100
	max_iter	400
	archive_size	100
	q	0.3
	xi	0.8
PSO	pop	100
	max_iter	400
	w	0.8
	c1	0.7
	c2	0.7

Table 4: Parameters used for comparing performance on the rastrigin function

3.1 Quality of Results

The global minima of the rastrigin function is at the origin and has a value of 0. Since the complexity of the problem task increases with different dimensionalities of the rastrigin function, we tested different settings. The results obtained for these complexity-levels can be seen in the table below:

Dimension	GA	ACO	PSO
5	2	15.1	1
10	6	71.1	11.9
15	16.9	121.7	66.7

Table 5: Minima found for the N-Dimensional Rastrigin Function Problem.

The experimental results in the table above show a clear degradation in optimization quality for all three algorithms as the dimensionality of the rastrigin function increases. This behavior is expected, since the number of local minima grows exponentially with dimension. While PSO performs best for the low-dimensional case, its performance decreases significantly with increasing dimensions and number of local minima. This indicates that PSO explores well at the beginning, but loses diversity too early and converges prematurely into a local minimum. The same holds true for ACO which performs worst across all tested settings. ACO in our case appears insufficient to escape the many local minima of the Rastrigin function. The performance may be slightly optimized by tuning the parameters even more, but this doesn't change the fact that ACO isn't suited for continuous problem-tasks. It performs better on combinatorial or discrete problems. The most consistent performer was GA. Although its error increases from 2.0 to 16.9 as the dimension rises from 5 to 15, this growth is moderate in comparison to PSO and ACO. The population-based recombination and mutation mechanisms seem to preserve diversity more effectively, making GA more robust to high dimensionality.

3.2 Runtime analysis

Since the number of iterations was fixed at 400 for all experiments, runtime differences reflect the per-iteration computational cost of each algorithm and how it scales with the dimensionality. PSO consistently achieves the lowest runtime (around 0.21s), and its runtime remains nearly constant across all dimensions tested. This indicates that PSO has the smallest per-iteration cost and that its simple update rules scale weakly with dimension. GA shows moderate runtimes, which increase gradually as the dimension increases. ACO is defenitely the slowest algorithm by a significant margin, and its runtime scales strongly with dimension.

Dimension	GA	ACO	PSO
5	0.38	0.72	0.21
10	0.47	1.22	0.21
15	0.53	1.74	0.22

Table 6: Runtimes for the Rastrigin Function Problem in Seconds.

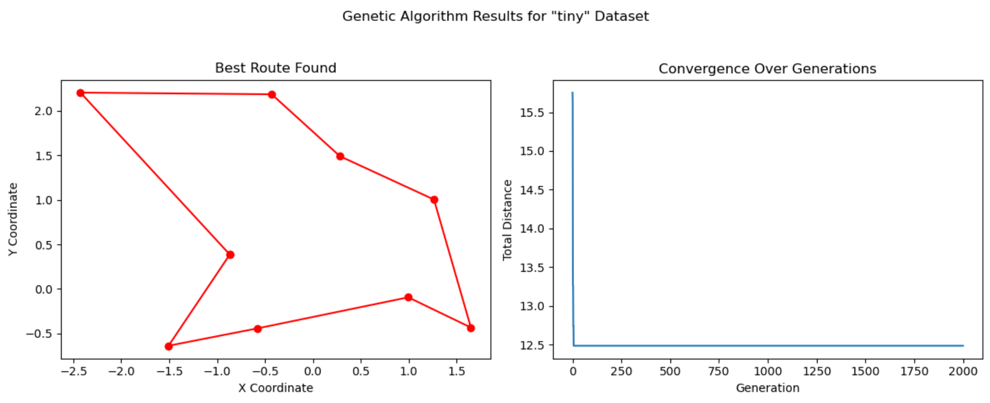
3.3 Conclusion

Across all dimensions, GA provides the best balance between the quality of the result and runtime. It scales moderately in runtime and maintains stable performance as dimension grows. PSO is by far the fastest method but it may fail and get stuck in local minima more easily at higher dimensions. ACO is both the slowest and least accurate under the tested settings. Overall, these experiments show that GA is the most robust, while PSO is the most efficient algorithm.

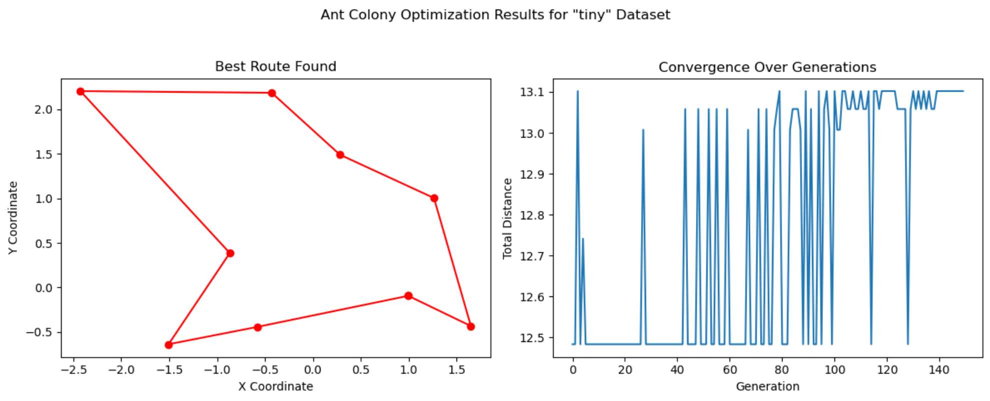
4 Discussion

The experiments highlight that different self-organizing techniques are suited to different types of optimization problems. Ant Colony Optimization (ACO) performs exceptionally well on the TSP, especially for large instances, where its pheromone-based search efficiently exploits the combinatorial structure of the problem. In contrast, the Genetic Algorithm (GA) clearly outperforms ACO and PSO on the continuous Rastrigin function and still delivers competitive results on the TSP. This suggests that GA is the most flexible and broadly applicable method among the three, as it handles both discrete and continuous problems reasonably well. ACO, on the other hand, appears highly effective but more specialized, excelling particularly on routing-like combinatorial tasks such as the TSP.

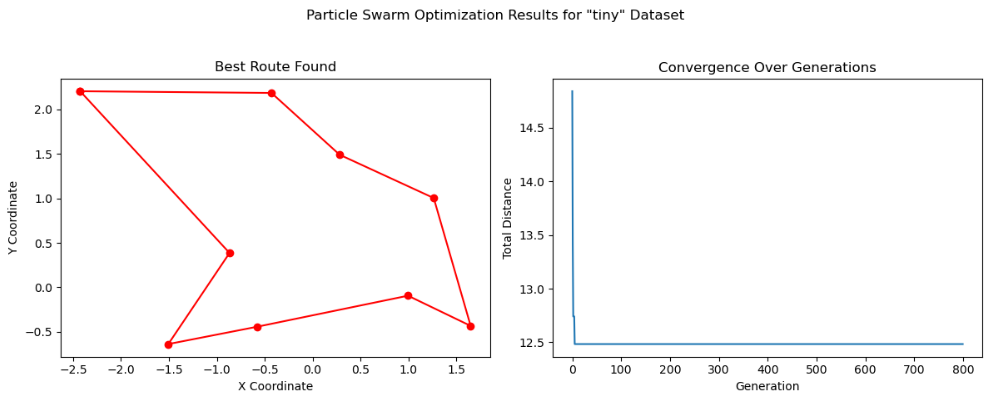
A TSP Algorithm Comparison Plots



(a) GA on Tiny Dataset

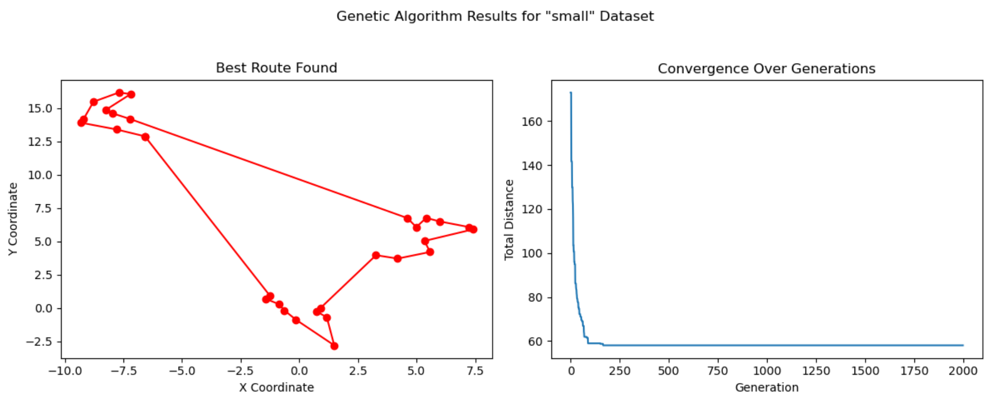


(b) ACO on Tiny Dataset

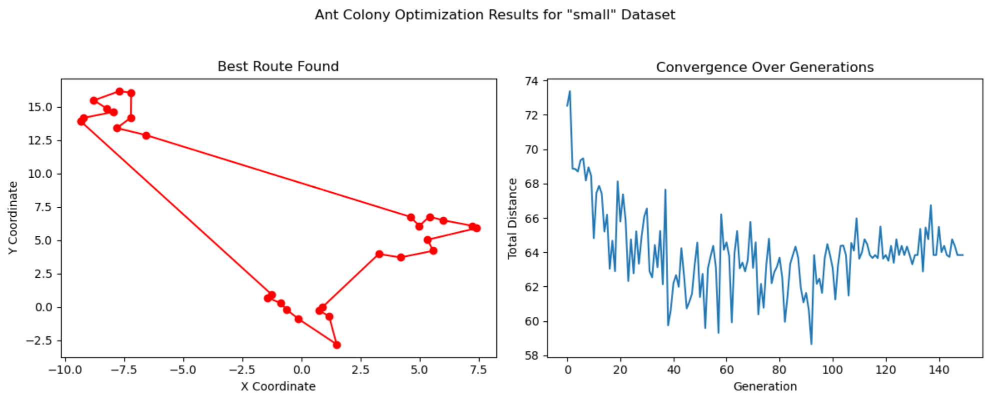


(c) PSO on Tiny Dataset

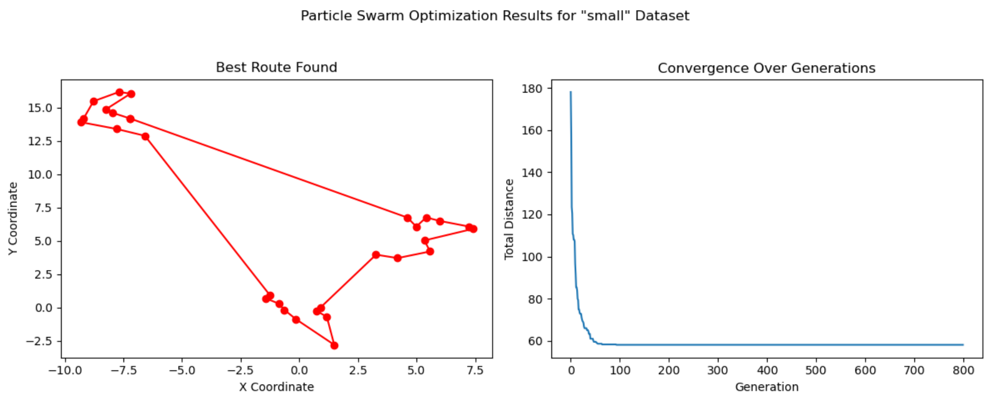
Figure 3: Comparison of GA, ACO, and PSO on the Tiny TSP dataset (10 cities).



(a) GA on Small Dataset

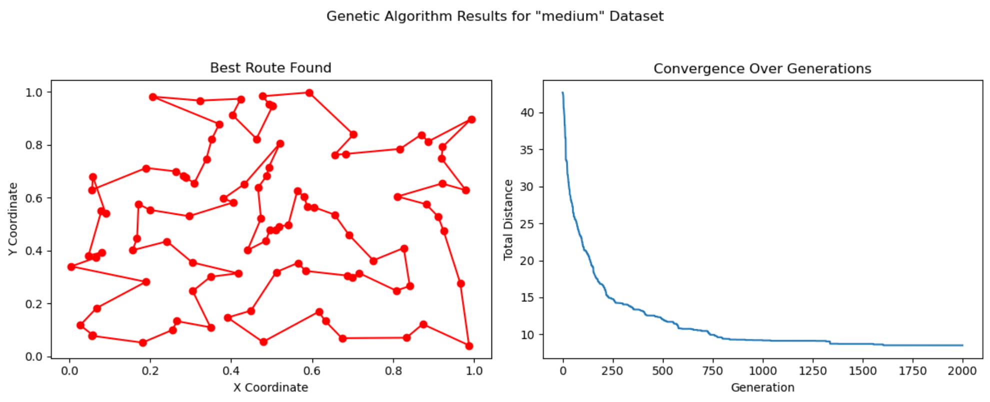


(b) ACO on Small Dataset

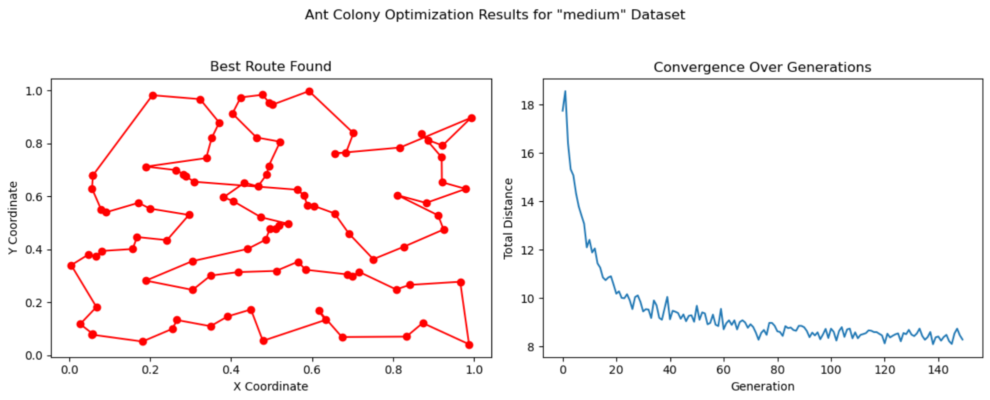


(c) PSO on Small Dataset

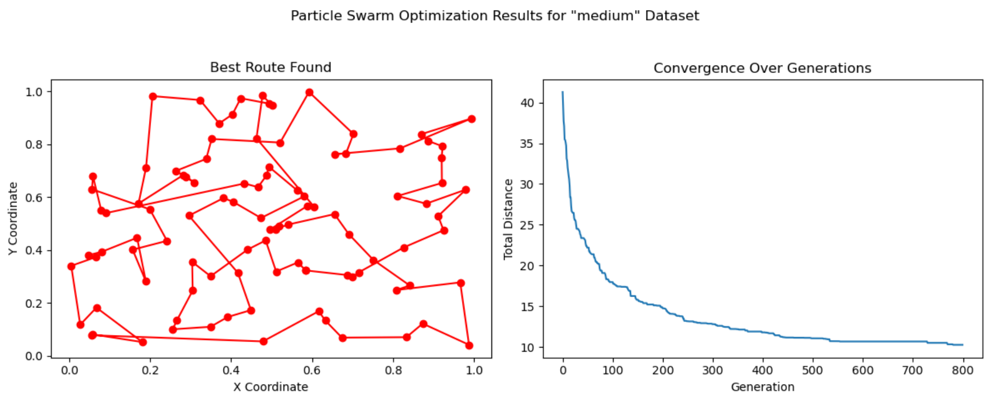
Figure 4: Comparison of GA, ACO, and PSO on the Small TSP dataset (30 cities).



(a) GA on Medium Dataset

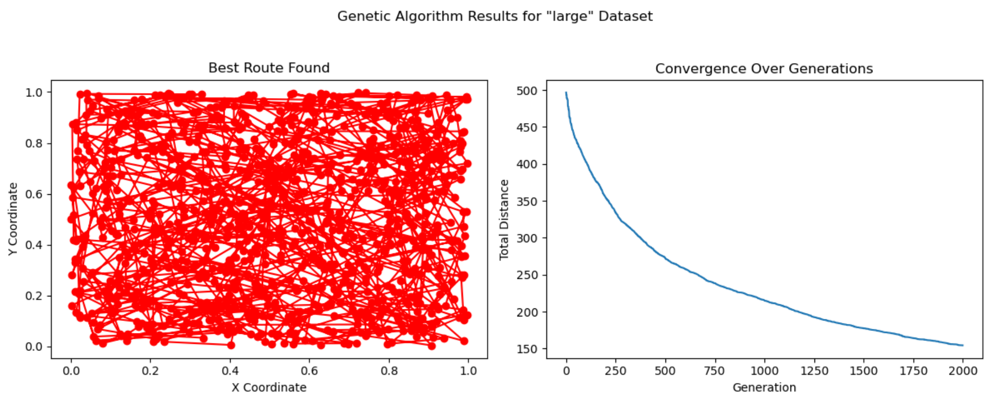


(b) ACO on Medium Dataset

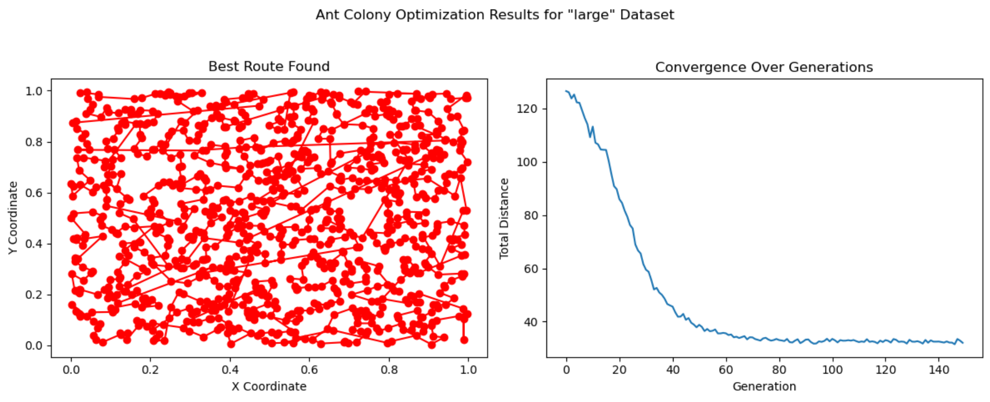


(c) PSO on Medium Dataset

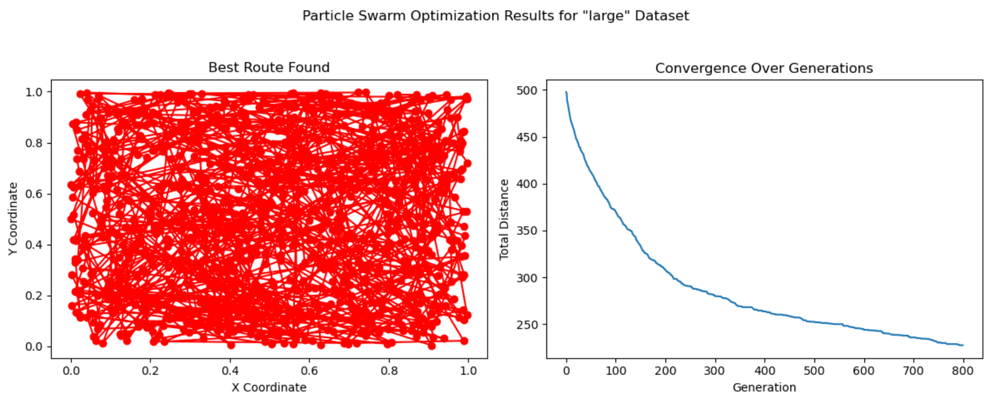
Figure 5: Comparison of GA, ACO, and PSO on the Medium TSP dataset (100 cities).



(a) GA on Large Dataset



(b) ACO on Large Dataset



(c) PSO on Large Dataset

Figure 6: Comparison of GA, ACO, and PSO on the Large TSP dataset (1000 cities).

References

[1] acu192. *Dataset TSP*. https://github.com/acu192/fun-tsp-challenge/tree/master. Accessed: 2025-11-25. License: MIT. 2025.

[2] Amir Omeradzic and Hans-Georg Beyer. "Progress analysis of a multi-recombinative evolution strategy on the highly multimodal Rastrigin function". In: *Theoretical Computer Science* 978 (2023), p. 114179. ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2023.114179. URL: https://www.sciencedirect.com/science/article/pii/S0304397523004929.

[3] guofei9987. *scikit-opt*. https://github.com/guofei9987/scikit-opt#. Accessed: 2025-11-25. License: MIT. 2025.

[4] tsadreas. ACO_R. https://github.com/tsadreas/ACO_R. Accessed: 2025-11-25. License: GPLv2. 2025.