Comparison of Genetic Algorithm and Simulated Annealing for Solving the Traveling Salesperson Problem

Gociu Radu

January 12, 2025

Abstract

This report presents a comprehensive comparative analysis of the Genetic Algorithm (GA) and Simulated Annealing (SA) for solving instances of the Traveling Salesperson Problem (TSP) from TSPLIB. GA employs a population-based approach with tournament selection, Partially Mapped Crossover (PMX), and *two-opt* mutation to iteratively refine solutions. SA uses a single-solution approach with a high initial temperature and an exponential cooling schedule to explore the solution space. The experimental results demonstrate that GA consistently outperforms SA in terms of solution quality, particularly for larger problem instances. GA's initialization and refinement steps enhance its ability to achieve near-optimal solutions, while SA's probabilistic acceptance mechanism leads to greater variability.

1 Introduction

The Traveling Salesperson Problem (TSP) is a well-known NP-hard problem in combinatorial optimization, with applications in logistics, manufacturing, and network design. The objective is to find the shortest possible route that visits each city exactly once and returns to the starting point. Due to its computational complexity, exact solutions become impractical for large instances, making metaheuristic algorithms such as Genetic Algorithms (GA) and Simulated Annealing (SA) essential tools for finding near-optimal solutions efficiently.

This report compares GA and SA, two popular metaheuristic approaches for solving TSP instances from TSPLIB. GA employs a population-based strategy involving selection, crossover, and mutation, combined with a *two-opt* local optimization to refine routes. SA, on the other hand, uses a single solution and explores the solution space probabilistically, controlled by a temperature-dependent acceptance function. Performance is evaluated across multiple TSPLIB instances, ranging from small examples like `berlin52` to large and complex instances like `usa13509`. The study provides insights into the strengths and

limitations of each algorithm in terms of scalability, convergence, and solution quality.

2 Methods

2.1 City Matrix and Distance Calculations

The distance matrix represents pairwise distances between all cities in the TSP instance. The matrix is constructed using the Euclidean distance formula:

$$d_{ij} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2} \tag{1}$$

Here, d_{ij} represents the distance between city i and city j. The matrix is symmetric, as $d_{ij} = d_{ji}$. The matrix is essential for fast lookups during route evaluations and is implemented using a 2D vector structure.

2.2 Genetic Algorithm (GA)

The Genetic Algorithm (GA) is a population-based optimization method inspired by natural selection. It iteratively refines a population of candidate solutions through selection, crossover, mutation, and local optimization.

2.2.1 Initial Population Generation

The initial population consists of random permutations of city indices, representing possible routes. Each chromosome is initialized as follows:

- **Size:** The size of the population depends on the problem instance (e.g., 50–200).
- **Diversity:** The initial generation is randomized to ensure exploration of the solution space. The city indices are shuffled using a random permutation generator.

This ensures that different parts of the search space are sampled, reducing the likelihood of premature convergence.

2.2.2 Selection Mechanism

Tournament selection with a group size of 5 was used. In each tournament, multiple individuals are randomly selected, and the one with the best fitness is chosen as a parent. This approach balances exploration and exploitation by giving high-quality solutions a higher probability of selection while maintaining genetic diversity.

2.2.3 Crossover (Recombination)

Partially Mapped Crossover (PMX) was employed to produce offspring. PMX swaps subsections between two parent routes and ensures that the offspring inherit valid permutations of cities. By preserving relative city positions, PMX helps create high-quality offspring that share desirable traits from both parents.

2.2.4 Mutation and Local Optimization

Two-opt mutation was applied to refine the routes. This operation swaps and reverses sub-sequences within the route to shorten it and improves the algorithm's ability to escape local optima.

2.3 Simulated Annealing (SA)

Simulated Annealing (SA) is a single-solution optimization approach inspired by the annealing process in metallurgy. It starts with a high initial temperature, which gradually cools down, allowing the algorithm to probabilistically accept worse solutions and escape local optima.

2.3.1 Initialization and Neighborhood Exploration

The initial solution is a random permutation of cities. The neighborhood is explored by applying a *two-opt* operation to reverse city subsequences and generate new routes.

2.3.2 Temperature and Cooling Schedule

The initial temperature was set to 10000.0, with an exponential cooling rate of 0.9996. This configuration allows the algorithm to accept worse solutions at the beginning and gradually focus on refining the route.

2.3.3 Acceptance Criterion

New solutions with a lower cost are always accepted. Worse solutions are accepted with a probability proportional to $e^{-\Delta E/T}$, where ΔE is the difference in cost and T is the current temperature. This probabilistic acceptance helps SA avoid premature convergence.

3 Results and Discussion

3.1 Results for the Easiest Instances

Table 1: Performance of GA and SA for the Easiest Instances

Instance	GA	SA	Best Solution	GA Deviation	SA Deviation
berlin52	7544.37	8009.87	7542	0.033%	6.20%
fl3795	28820.86	30951.93	28772	0.17%	6.23%
d1655	62936.96	68306.22	62128	1.30%	9.94%
pr1002	262150.49	276453.41	259045	1.20%	6.73%

Interpretation: GA achieves near-optimal results for easy instances, while SA shows larger deviations due to its reliance on probabilistic acceptance. For smaller instances like `berlin52`, the computational cost is low, allowing the GA to explore efficiently, while SA's performance is constrained by its exploration mechanism.

3.2 Results for Medium Difficulty Instances

Table 2: Performance of GA and SA for Medium Difficulty Instances

Instance	GA	SA	Best Solution	GA Deviation	SA Deviation
pr2392	385030.40	408299.90	378032	1.85%	8.00%
pcb3038	142209.28	150657.60	137694	3.28%	9.39%
dsj1000	18847772.76	20335480.38	18659688	1.01%	8.97%

Interpretation: GA shows low deviations for medium instances, whereas SA's performance varies significantly, particularly for larger instances like `dsj1000`. The larger the problem size, the more challenging it becomes for SA to maintain consistent performance due to its single-solution approach.

3.3 Results for the Hardest Instances

Table 3: Performance of GA and SA for the Hardest Instances

Instance	GA	SA	Best Solution	GA Deviation	SA Deviation
rl5915	580410.53	613696.98	565530	2.63%	8.52%
rl11849	980752.38	1000876.15	923288	6.23%	8.41%
usa13509	21127735.94	21584531.66	19982859	5.73%	8.02%

Interpretation: GA significantly outperforms SA for the hardest instances, achieving lower deviations and more stable performance. Instances like `usa13509`

demonstrate the advantages of GA's population-based approach and local optimization, while SA struggles with consistent improvement due to its reliance on a single solution.

3.4 Comparison of GA and SA

The results indicate that GA generally outperforms SA across all instances, particularly for larger and more complex problem sizes. GA benefits from its population-based approach, which maintains diversity and refines solutions locally using *two-opt* mutation. In contrast, SA's single-solution exploration and reliance on probabilistic acceptance lead to less consistent outcomes.

Key observations include:

- GA achieves consistently lower deviations from the best-known solutions.
- SA performs adequately for small instances but degrades as problem size increases.
- GA's population refresh mechanism helps prevent stagnation.
- SA's acceptance criterion allows some exploration but cannot match GA's efficiency for large instances.

4 Conclusions

The Genetic Algorithm demonstrates superior performance in solving TSP instances compared to Simulated Annealing, particularly for larger and more complex cases. GA's combination of selection, crossover, mutation, and local optimization allows it to explore the solution space efficiently and refine routes to near-optimal levels. By leveraging population diversity and applying *two-opt* improvements, GA maintains robustness and avoids premature convergence.

Simulated Annealing, while effective for small to medium instances, struggles with larger TSP problems due to its reliance on a single solution path and probabilistic transitions. Future research could focus on hybrid approaches that combine the strengths of GA and SA to further enhance performance.

5 References

- TSPLIB Documentation: `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/`.
- Mitchell, M. *An Introduction to Genetic Algorithms.*
- Goldberg, D.E. *Genetic Algorithms in Search, Optimization, and Machine Learning.*
- Michalewicz, Z. *Genetic Algorithms + Data Structures = Evolution Programs.*

- Holland, J.H. *Adaptation in Natural and Artificial Systems.*
- Shredderzwj's GitHub Repository for TSPLIB Instances: `https://github.com/shredderzwj/TSPLIB/blob/master/res/pr1002.tsp`.
- YouTube Video on Metaheuristic Algorithms: `https://youtu.be/XaXsJJh-Q5Y?si=2SOpLDyGM9oufMfq`.
- Dorigo, M. *Ant Colony Optimization.*