Comparative Analysis of Genetic Algorithm and Simulated Annealing in Solving the Traveling Salesman Problem

Popa Andrei, Octavian-Ștefan Regatun

January 10, 2024

Abstract

This study aims to make an analytic and comprehensive comparison of two heuristic algorithms, Genetic Algorithm (GA) and Simulated Annealing (SA), applied to the Traveling Salesman Problem (TSP), providing insights into the strengths and limitations of these algorithms in solving TSP. The TSP, a well-known combinatorial optimization challenge, requires finding the shortest possible route that visits a set of cities and returns to the origin city. The objective was to evaluate the effectiveness of GA and SA in identifying optimal or near-optimal solutions for TSP instances of varying complexity. The findings reveal that while GA is advantageous in achieving high-quality solutions, ACO (Ant Colony Optimization), as part of the GA implementation, shows superior performance in terms of computational speed. Notably, SA tends to provide better solutions in terms of closeness to the optimal, but it is slower and requires more computational time compared to GA.

1 Introduction

The field of combinatorial optimization encompasses a range of problems with significant practical applications, among which the Traveling Salesman Problem (TSP) is notably prominent. The TSP poses a challenge of finding the most efficient route to visit a set of cities and return to the starting point, a task that has implications in logistics, planning, and numerous other domains. This paper focuses on the application of two heuristic algorithms, Genetic Algorithm (GA) and Simulated Annealing (SA), to solve TSP instances. Previous studies have independently highlighted the capabilities of GA and SA in various optimization scenarios. However, there is a gap in the literature regarding a direct, comprehensive comparison of these methods applied specifically to TSP. We address this gap by implementing both algorithms and testing them on multiple TSP instances, ranging from simpler to more complex scenarios. Our approach involves a detailed examination of each algorithm's performance, considering factors like solution quality, computational efficiency, and scalability. The findings aim to offer a nuanced understanding of the relative strengths of GA and SA in tackling the TSP, providing valuable guidance for practitioners and researchers in selecting the most suitable heuristic method for specific optimization challenges.

We will establish the next notations that will be used throughout the article:

SA = Simulated Annealing

GA = Genetic Algorithm

TSP = Travelling Salesman Problem

2 Methods

2.1 Genetic Algorithm Implementation for TSP

The Genetic Algorithm (GA) implementation for TSP is designed to optimize the route selection by simulating evolutionary processes. Key components of this implementation include:

- *Chromosome Representation*: Each chromosome represents a potential solution to the TSP, encoded as a sequence of city indices indicating the order of visits.
- *Fitness Function*: The fitness of a chromosome is inversely proportional to the total travel distance of the represented route. Lower distances correspond to higher fitness values.
- *Selection Mechanism*: A tournament selection approach is employed, where a subset of the population competes, and the fittest individuals are selected for breeding.
- *Crossover and Mutation*: We use ordered crossover to preserve the sequence of cities without duplication, and swap mutation to introduce variability.
- *Population Management*: The population evolves over generations, with an method inspired from nature names "Ant Colony Optimization".

Integration of Ant Colony Optimization in Genetic Algorithm: Ant Colony Optimization (ACO), inspired by the foraging behavior of ants, is integrated into our Genetic Algorithm to enhance the exploration and exploitation capabilities in solving TSP. ACO's logic stems from how real ants find the shortest paths between their colony and food sources.

Pheromone Trails: In ACO, solutions are related to the paths taken by ants. Ants communicate and influence each other's paths through pheromone trails. Higher pheromone concentrations on a path signal a more desirable route. In our implementation, this concept is used to influence the selection of routes in the genetic algorithm's crossover and mutation processes.

Probabilistic Path Selection: Ants probabilistically choose their paths based on the intensity of pheromones, which allows exploration of new paths while exploiting known good paths. This behavior is mirrored in the GA to balance exploration of the search space with the exploitation of known good solutions.

Pheromone Evaporation: To avoid convergence to a suboptimal path, pheromone trails evaporate over time, reducing their influence. This mechanism is integrated into our GA to ensure diversity in the population and to prevent premature convergence.

Algorithm Integration: The ACO is integrated into the GA as follows:

- *Initial Population*: The initial population of routes in the GA is influenced by simulated ant trails, ensuring a diverse set of starting solutions.
- *Fitness Evaluation*: The fitness of each route in the GA is augmented by considering the pheromone levels, akin to the desirability of paths in ACO.
- *Crossover and Mutation*: These GA operations are guided by pheromone information, promoting the inheritance of traits from better-performing solutions.

This hybrid approach leverages the strengths of both GA and ACO, utilizing GA's robust search capabilities and ACO's dynamic adaptation to changing environments, making it particularly effective for complex TSP instances.

Optimization - Dynamic Mutation Rate:

A key factor in the success of Genetic Algorithms (GA) is the delicate balance between exploration and exploitation, which is critically influenced by the mutation rate. In our GA implementation for

the TSP, we adopt a dynamic mutation rate strategy to enhance this balance, adapting the rate of mutation as the algorithm progresses through generations.

In the context of GA, exploration refers to the ability of the algorithm to search through a wide range of the solution space, thus avoiding local optima, while exploitation focuses on intensifying the search around promising areas to refine solutions. A static mutation rate might lead either to excessive randomness (over-exploration) or rapid convergence to suboptimal solutions (over-exploitation).

The implemented dynamic mutation rate is designed to change adaptively with the number of generations. Initially, a higher mutation rate encourages exploration, allowing the algorithm to search through diverse regions of the solution space. As the algorithm approaches the maximum number of generations, the mutation rate decreases, shifting the focus towards exploitation. This gradual reduction in the mutation rate allows the algorithm to fine-tune the solutions, enhancing the chances of converging to the global optimum.

This approach provides a self-adjusting mechanism that responds to the algorithm's stage of evolution. In the early stages, where the risk of convergence to local optima is high, it promotes genetic diversity. In later stages, as the algorithm begins to hone in on potential solutions, it reduces disruptive mutations, thereby refining the search. Consequently, the dynamic mutation rate plays a pivotal role in enhancing the GA's efficiency and effectiveness in solving complex problems like TSP.

Optimization - Optimized Crossover Method in Genetic Algorithm: The crossover mechanism is a fundamental genetic operator in GAs, responsible for combining the genetic information of two parent solutions to produce new offspring. In our GA implementation for the TSP, we have adopted an optimized crossover method that enhances the diversity and quality of the solutions.

Our crossover method starts by deciding whether to perform crossover based on a predefined crossover rate (in this case, 0.7). If crossover is not performed, one of the parent genes is randomly selected to pass to the next generation. This initial step introduces an additional layer of randomness, preventing the algorithm from being overly deterministic and ensuring diversity in the population.

When crossover is performed, a segment of the route is selected from each parent (between two randomly chosen points). Each position within the selected segment is then filled with a gene from either parent, chosen randomly. This approach ensures that each offspring receives a blend of traits from both parents, promoting beneficial traits' propagation.

After the initial segment is filled, the rest of the offspring's route is completed by iterating over the second parent's genes. If a gene (city) is not already present in the offspring (ensured by a set), it fills the next available placeholder. This step guarantees that each city appears exactly once in the offspring, maintaining the validity of the TSP solution.

Advantages of Optimized Crossover:

- *Balanced Exploration and Exploitation*: By blending genes from both parents and introducing randomness, the crossover operator effectively explores new areas of the solution space while exploiting the advantageous traits of the parents.
- *Diversity Preservation*: The method helps maintain genetic diversity within the population, crucial for avoiding premature convergence.
- *Solution Quality*: This approach enhances the likelihood of generating high-quality offspring, contributing to the overall effectiveness of the GA in finding optimal solutions for TSP.

In summary, the optimized crossover method in our GA implementation plays a pivotal role in the algorithm's ability to efficiently and effectively navigate the complex solution space of the TSP.

Algorithm Parameters: Parameters such as population size, mutation rate, number of generations, and the initialization of the ant colony(the initialization of chromosomes) were fine-tuned based on preliminary experiments to balance exploration and exploitation.

2.2 Simulated Annealing Implementation for TSP

Simulated Annealing is a probabilistic technique that mimics the process of heating a material and then slowly lowering the temperature to decrease defects, thus minimizing the system's energy. In the context of TSP, it is used to find a path that has the minimum total distance.

- *Initial Solution and Temperature*: The algorithm starts with a randomly generated route and a high initial temperature.
- *Neighbor Solution Generation*: A new solution is generated by slightly modifying the current route, typically by swapping two cities.
- *Acceptance Probability*: The acceptance of a new solution, especially if it's worse than the current one, is based on a probability function that depends on the difference in tour lengths and the current temperature.
- *Cooling Schedule*: The temperature is gradually decreased according to a predetermined schedule, reducing the likelihood of accepting worse solutions over time.

Algorithm Parameters: Key parameters like initial temperature, cooling rate, and termination criterion were calibrated to optimize the performance on TSP instances.

- *Initial Temperature*: Set to a high value of 1,000,000,000. This high starting temperature allows the algorithm to explore a wide range of solutions initially, including accepting sub-optimal paths, to avoid local minima.
- *Cooling Rate*: The cooling factor is initially set to 0.5. This rate determines how quickly the temperature decreases, balancing between exploration and exploitation. As the algorithm progresses, the cooling rate is adaptively adjusted based on the progress of the solution's quality.
- *Reheating Mechanism*: If no improvement is observed for 100 iterations, the temperature is increased by a factor of 2. This reheating helps to escape local minima by allowing the algorithm to explore more widely again.
- *Termination Criterion*: The algorithm employs a dynamic stopping criterion, terminating when there is no significant improvement in the route for 10,000 consecutive iterations. This criterion helps in concluding the search once it is unlikely to find a better solution, thereby optimizing computational resources.

2.3 Test Instances and Experiment Setup

For this study, a diverse set of TSP instances were selected from established libraries, ensuring a range from simpler to more challenging routes. The chosen instances reflect varying numbers of cities and different topological complexities. Each algorithm was run on these instances under similar computational conditions to ensure a fair comparison. The performance metrics included solution quality, computational time, and scalability.

Evaluation Criteria: The algorithms were evaluated based on their ability to find the shortest possible route within a reasonable computation time, with a focus on their performance in larger, more complex instances.

3 Experiment

Table 1: Results of Ant Colony Optimization over Multiple Iterations

Name	Size	Average Len.	Best Len.	Std. Dev.	Average Time (s)	Optimal Len.
rl5934	5934	664972.67	618964	5908.70	2261.61	556045
pr2392	2392	456132.3	391521.1	2855.62	148.6561	378032
u1817	1817	65478.71	64082.61	652.51	65.4407	57201
nrw1379	1379	68767.15	67827.0	566.65	28.7963	56638
pr1002	1002	317520.2	310337	3829.14	11.6954	259045
rat783	783	9626.85	9362.9	107.55	11.3333	8806
pr439	439	131424.2	127521	2083.97	1.1026	107217
lin318	318	48359.85	47732	361.84	22.2095	42029
st70	70	703.87	695.84	5.26	1.4776	675
berlin52	52	8117.99	8055.95	8117.99	0.9474	7542

Table 2: Results of Simulated Annealing over Multiple Iterations

Name	Size	Average Len.	Best Len.	Std. Dev.	Average Time (s)	Optimal Len.
rl5934	5934	5809510.48	5499618.83	147717.37	13452.88	556045
pr2392	2392	378062.84	378062.80	0.022	5764.66	378032
u1817	1817	72104.76	70754.22	700.27	4434.58	57201
nrw1379	1379	141958.21	135211.72	3840.85	2456.10	56638
pr1002	1002	349193.52	342639.29	2663.86	1998.83	259045
rat783	783	19469.01	19347.81	104.60	1686.57	8806
pr439	439	166238.42	158844.87	4257.08	743.64	107217
lin318	318	58837.28	55982.14	1859.91	597.11	42029
st70	70	698.31	680.96	10.78	195.54	675
berlin52	52	7743.10	7544.37	123.15	132.69	7542

4 Conclusion

This study embarked on a comprehensive evaluation of two heuristic optimization methods: Genetic Algorithms (GA) and Simulated Annealing (SA), applied to the Traveling Salesman Problem (TSP). Through a series of designed experiments, we have demonstrated that both algorithms exhibit distinct strengths under different conditions.

The GA, with its dynamic mutation rate and optimized crossover methods, showed remarkable performance in maintaining diversity within the population and preventing premature convergence. The integration of Ant Colony Optimization into the GA further reinforced its exploratory capabilities, enabling the discovery of near-optimal solutions with notable consistency. On the other hand, SA demonstrated robustness in its ability to escape local optima, thanks to its probabilistic acceptance of solutions and adaptive cooling schedule.

Our findings indicate that the choice between GA and SA for solving TSP should be guided by specific problem parameters and performance criteria. For instances where solution quality is paramount, and computational resources are ample, GA stands out as the preferred choice. In scenarios where time is of the essence, SA might be advantageous due to its typically faster convergence.

Future research could explore the potential of hybrid algorithms that combine the exploratory mechanisms of GA with the fine-tuning abilities of SA. Additionally, further studies into adaptive parameter tuning could yield algorithms that are more resilient to the varying complexities of TSP instances.

Conclusively, GA stands out in terms of speed and computational efficiency, making it a preferable choice for scenarios where time is a critical factor. However, if the primary goal is the quality of the solution, particularly for complex and larger instances, Simulated Annealing may be more effective, despite its longer computation time. The choice between these two algorithms should therefore be guided by the specific requirements of the problem at hand — whether speed is prioritized over solution accuracy, or vice versa. Future research could focus on hybrid approaches that combine the exploratory efficiency of GA with the exhaustive search capabilities of SA, potentially leading to algorithms that are both fast and yield high-quality solutions.

[1] Bäck, T., Schütz, M. (1996). *Intelligent mutation rate control in canonical genetic algorithms*. In: Raś, Z.W., Michalewicz, M. (eds) Foundations of Intelligent Systems. ISMIS 1996. Lecture Notes in Computer Science, vol 1079. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-61286-6_141

[2] *LaTeX Tutorial*. https://www.overleaf.com/learn/latex/Tutorials/

[3] MIT OpenCourseWare. *Gradient Descent, Step Size, Rate of Convergence*. YouTube.

[4] GeeksforGeeks. *ML - Stochastic Gradient Descent (SGD)*.

[5] James Brookhouse and Alex Freitas. *Fair Feature Selection: A Comparison of Multi-Objective Genetic Algorithms*. (2023). https://arxiv.org/pdf/2310.02752.

[6] Krutika Sarode and Shashidhar Reddy Javaji. *Hybrid Genetic Algorithm and Hill Climbing Optimization for the Neural Network*. (2023). https://arxiv.org/pdf/2308.13099.

[7] Alexander E. I. Brownlee and others. *Enhancing Genetic Improvement Mutations Using Large Language Models*. (2023). https://arxiv.org/pdf/2310.19813.

[8] Tarek Faycal and Claudio Zito. *Direct Mutation and Crossover in Genetic Algorithms Applied to Reinforcement Learning Tasks*. (2022). https://arxiv.org/pdf/2201.04815.

[9] Endah Rokhmati Merdika Putri and others. *A Deep-Genetic Algorithm (Deep-GA) Approach for High-Dimensional Nonlinear Parabolic Partial Differential Equations*. (2023). https://arxiv.org/pdf/2311.11558.

[10] *Genetic Algorithm Course on Udemy*. https://www.udemy.com/course/geneticalgorithm/?utm_source=adwords&utm_medium=udemyads&utm_campaign=LongTail-New_la.EN_cc.ROWMTA-B&utm_content=deal4584&utm_term=_._ag_101378276820_._ad_533999945410_._kw__._de_c_._dm__._pl__._ti_dsa-1007766171032_._li_1011828_._pd__._&matchtype=&gad_source=1&gclid=CjwKCAiAvJarBhA1EiwAGgZl0GvWMnxn0VmAxJwqE4VV7d56BwE. (Accessed: November 25, 2023).

[11] *Genetic Algorithms Course Website*. https://profs.info.uaic.ro/~eugennc/teaching/ga/.

[12] TheProjectSpot. Ant Colony Optimization for Hackers. https://www.theprojectspot.com/tutorial-post/ant-colony-optimization-for-hackers/10, Accessed on: Jan 9, 2024.

[13] The Coding Train. Introduction to Ant Colony Optimization. https://www.youtube.com/watch?v=BAejnwN4Ccw, Accessed on: Jan 9, 2024.

[14] Auctux. Ant Colony Optimization Explained. https://www.youtube.com/watch?v=Sk9QQUGMdY8, Accessed on: Jan 9, 2024.