Comparison of TSP Solutions on TSPLIB Data

Comparison of TSP Solutions on TSPLIB Data **1**
- Introduction 2
- Algorithm Descriptions 2
- Data Segmentation and Metrics 3
- Results and Analysis 3
  - Solution Cost Trends 3
  - Execution Time and Time Complexity 4
  - Accuracy and Deviation from Optimality 5
- Discussion and Insights 6
- Future Work 9
- Conclusion 9

Introduction

The **Traveling Salesman Problem (TSP)** is a classic optimization problem with numerous applications in logistics, telecommunications, and manufacturing. The objective is to determine the shortest possible route that visits all cities exactly once before returning to the starting point. Due to its NP-hard nature, solving the TSP becomes computationally expensive as the number of cities increases. This report presents a comparative analysis of several algorithms on TSP datasets from **TSPLIB**, focusing on solution quality, runtime, and scalability. The dataset includes instances with vertex counts ranging from 14 to 1000, allowing for a thorough examination of each algorithm's performance across varying scales.

Algorithm Descriptions

We analyzed six algorithms along with several variants for some, each suited for different problem scales and constraints:

1. **Held-Karp (Dynamic Programming)**: This algorithm provides an optimal solution by breaking down the problem into smaller subproblems using memoization. However, its exponential time complexity $O(2^N N^2)$ limits practical usage to instances with fewer than 20 vertices.
2. **Greedy Algorithm**: A constructive heuristic that iteratively selects the nearest unvisited city. While computationally efficient ($O(N^2)$), it often produces suboptimal solutions due to its local, myopic decision-making, especially as problem size increases.
3. **Simulated Annealing (SA)**: A probabilistic heuristic that explores the solution space broadly, allowing non-improving moves initially to escape local minima. We analyzed multiple SA variants that incorporate constructive initial solutions and local search optimizations (e.g., 2-opt or 3-opt), improving solution quality and convergence speed. While SA handles large instances, its effectiveness depends heavily on parameter tuning, with no guarantee of optimality.
4. **Genetic Algorithm (GA)**: A population-based heuristic that evolves solutions through selection, crossover, and mutation. The Memetic GA variant includes local search (2-opt or 3-opt) after crossover, which refines individuals within each generation. This approach enhances convergence and solution quality, particularly for medium-sized instances, though GA becomes computationally expensive for large graphs.
5. **Lin-Kernighan (LK)**: A powerful heuristic that dynamically selects kkk-opt moves to iteratively improve the tour, achieving near-optimal results for small to medium graphs. However, it can become computationally intensive for larger instances due to the complexity of evaluating potential moves.
6. **Christofides Algorithm**: An approximation algorithm for symmetric TSP with a guaranteed upper bound of 1.5 times the optimal solution, assuming the triangle inequality holds. While efficient ($O(N^3)$), it may underperform on non-metric or asymmetric problems.

Data Segmentation and Metrics

To manage the wide range of vertex counts ($10 \leq N \leq 1000$), we segmented the data into three ranges:

- **Small-Scale**: $10 \leq N \leq 50$
- **Medium-Scale**: $50 < N \leq 200$
- **Large-Scale**: $200 < N \leq 1000$

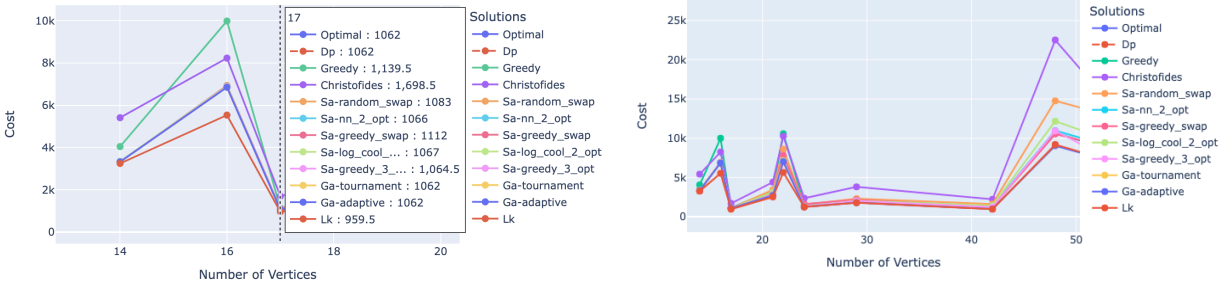
The key metrics are:

- **Solution Cost**: Total tour length for each solution.
- **Execution Time**: Time taken by each algorithm to compute a solution.
- **Deviation from Optimality**: Accuracy measured against optimal or best-known values.

Results and Analysis

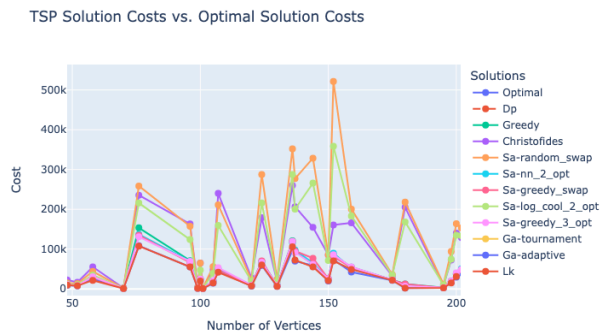
Solution Cost Trends

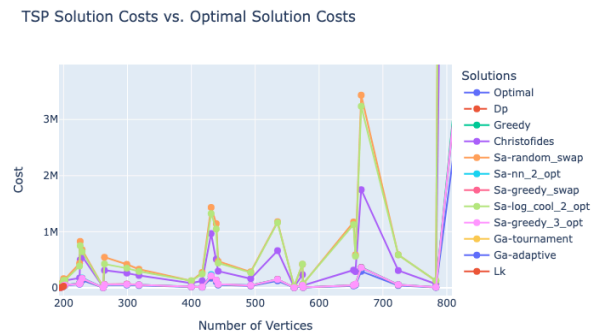
Small-Scale ($10 \leq N \leq 50$):



Dynamic Programming (DP), Greedy, Genetic Algorithm (GA) and Lin-Kernighan (LK) and most Simulated Annealing (SA) variants performed closely to the optimal solution. Christofides and SA Random Swap showed increased deviation as vertex count approached 50.

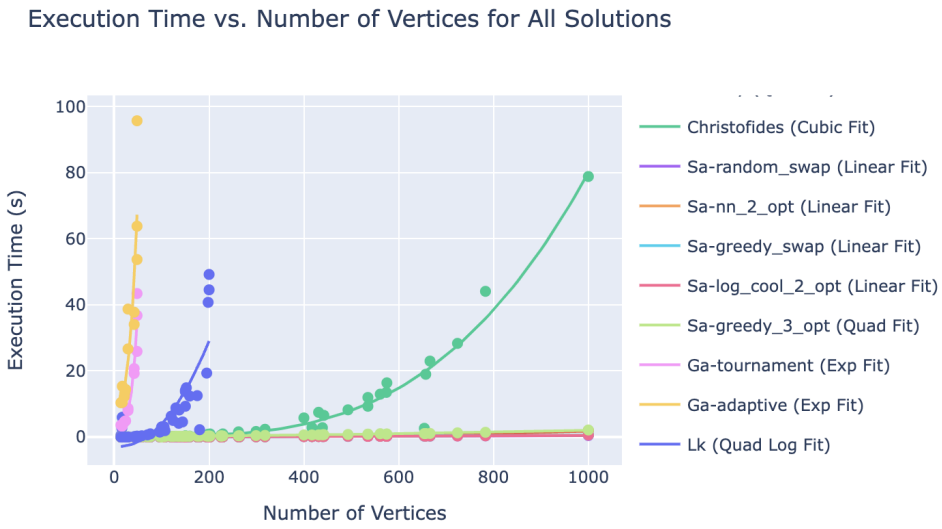
Medium-Scale ($50 < N \leq 200$), Large-Scale ($200 < N \leq 800$):



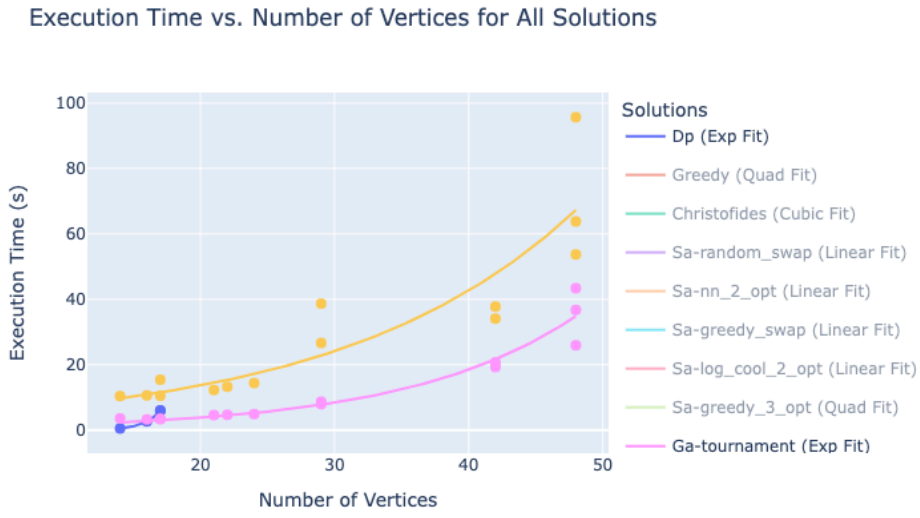


In both ranges, we are unable to generate solutions in practice for DP and GA variants due to exponentially increasing runtimes. Greedy begins to show significant deviation from the optimal solution as the vertex count increases. Christofides, SA Random Swap and SA Logarithmic Cooling also perform worse, especially beyond 75 vertices, indicating difficulty with larger problem sizes. The other SA variants and LK continue to perform reasonably well, but the latter (LK) is unable generate solutions beyond 200 vertices due to its quadratic logarithmic runtime.

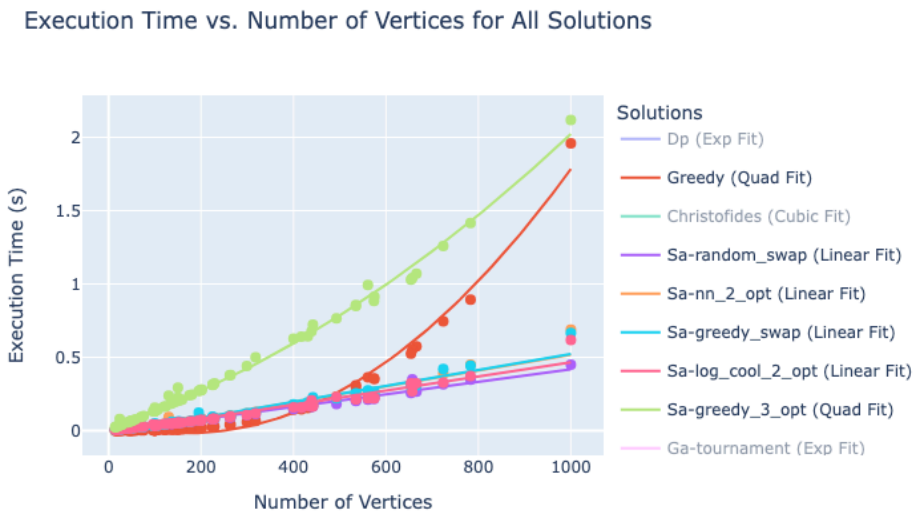
Execution Time and Time Complexity



GA Variants: Exhibits exponential growth in runtime, becoming infeasible for graphs (N > 50).
LK: Exhibits quadratic logarithmic growth in runtime, becoming infeasible for graphs (N > 200).
Christofides: Exhibits cubic growth in runtime, becoming infeasible for graphs (N > 400).



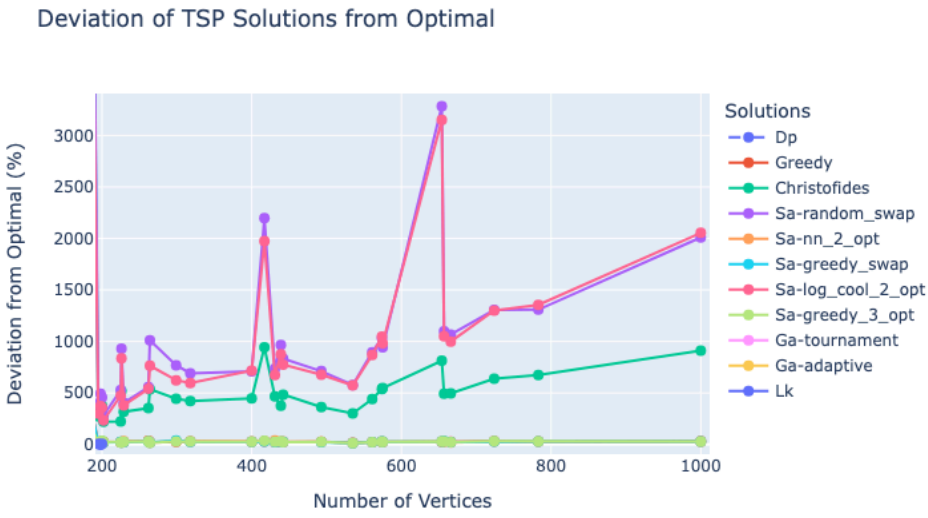
DP: The exponential growth of Held-Karp's execution time renders it impractical for N > 20.



Greedy and SA Variants: Both maintain quadratic/linear time growth, making them more scalable. SA is generally slightly faster than Greedy (except SA Greedy 3-opt) due to its probabilistic nature and offers flexibility in large instances.

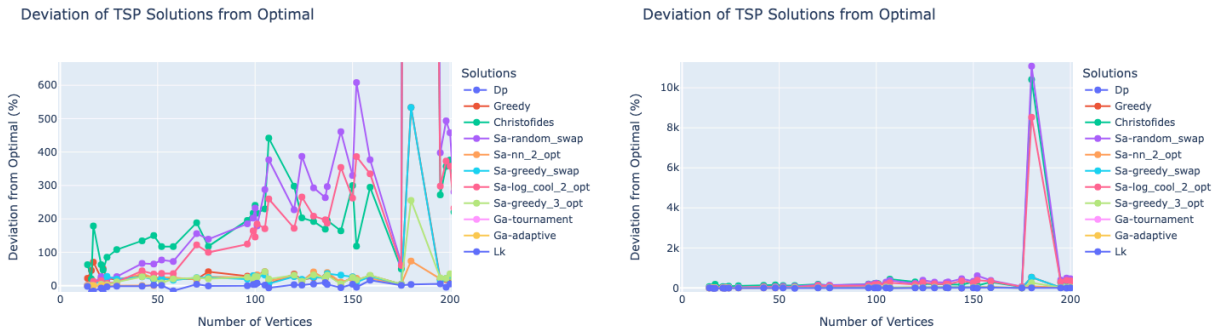
Accuracy and Deviation from Optimality

Large-Scale (200 < N ≤ 1000):



The highest deviations are seen in Christofides, SA Random Swap and SA Logarithmic Cooling, especially at 400 and 700 vertices. The other SA variants and Greedy occasionally exhibit spikes in deviation, but maintain acceptable performance (consistently under 50% deviation from optimal).

Small-Scale (10 ≤ N ≤ 50), Medium-Scale (50 < N ≤ 200):



Christofides, SA Random Swap and SA Logarithmic Cooling exhbit large deviations, even at smaller scales. Other SA variants show less deviations, but spikes are still observed, especially past 100 vertices. DP provides exact solutions for up to 20 vertices. LK and GA variants remain under 10% deviation for its explored range (N < 200). Greedy remains close to optimal but begins to deviate at larger scales.

Discussion and Insights

For this section, we summarize the performance of each algorithm based on three key factors: **Trade-offs Between Accuracy and Speed**, **Scalability**, and **Real-World Applicability**. These factors highlight the unique strengths and limitations of each algorithm in the context of solving the Traveling Salesman Problem (TSP).

- **Trade-offs Between Accuracy and Speed**: This factor examines the balance each algorithm strikes between solution accuracy and computational efficiency. Some algorithms guarantee optimal solutions but are computationally intensive, while others provide faster, approximate solutions with potential sacrifices in accuracy.
- **Scalability**: Scalability considers how well each algorithm can handle increasing problem sizes, particularly as the number of vertices grows. This is crucial for TSP applications that involve larger datasets, where certain algorithms become impractical due to their computational complexity.
- **Real-World Applicability**: Real-world applicability assesses the types of scenarios where each algorithm is most useful, taking into account the trade-offs in speed, accuracy, and scalability. Some algorithms are better suited for exact solutions on small datasets, while others offer approximate solutions that scale effectively for larger, more complex problems.

Algorithm Name	**Trade-offs Between Accuracy and Speed**	**Scalability**	**Real-World Applicability**
Held-Karp (DP)	Guarantees optimal solution but has exponential time	Limited to small problems due to exponential	Suitable for small-scale problems where exact solutions are required,

	complexity, making it impractical for larger graphs (typically limited to N < 20).	complexity.	such as logistics optimization with very few locations.
Greedy	Computationally efficient $O(N^2)$ but often produces suboptimal solutions due to its locally optimal choices.	Scales well but accuracy decreases with larger graphs.	Useful in real-time or embedded systems where fast, approximate solutions are acceptable, e.g., routing with minimal computational resources.
Christofides	Provides an approximation ratio of 1.5, trading some accuracy for scalability. Performs well if the triangle inequality holds but may severely underperform on non-metric or asymmetric graphs.	Scales reasonably well $O(N^3)$ but may struggle with non-metric problems and large graphs.	Effective in applications where the triangle inequality holds, such as certain logistics and delivery problems with symmetric distances.
SA Random Swap	Can find diverse solutions by allowing random swaps; accuracy varies, and runtime depends on cooling parameters.	Scales better than exact algorithms, but solution quality varies with parameter settings and is not guaranteed to be near-optimal.	Applicable when flexibility in accuracy is acceptable, e.g., exploratory scheduling or logistics optimization with time constraints.
SA Nearest Neighbor 2-opt	Starting with a nearest-neighbor solution and applying 2-opt moves improves accuracy and convergence speed, though solution quality is still parameter-dependent.	Scales well with larger instances but requires careful parameter tuning for optimal balance between accuracy and runtime.	Suitable for routing and scheduling applications where moderately accurate solutions are needed quickly, such as vehicle routing with fixed time windows.
SA Greedy Swap	Begins with a Greedy solution, improving accuracy over random initialization; combines Greedy efficiency with SA flexibility but depends on the cooling schedule for convergence.	Scales effectively, providing better initial solutions but still parameter-sensitive and heuristic in nature.	Useful for large, moderately complex problems where a good initial solution is beneficial, such as facility layout or path planning.

SA Logarithmic Cooling	Logarithmic cooling provides a gradual temperature decrease, improving solution quality by allowing longer exploration at higher temperatures, though it increases runtime compared to linear cooling.	Scales reasonably with improved accuracy for larger graphs, but runtime increases due to slower cooling.	Appropriate for applications requiring better-quality solutions with a relaxed runtime constraint, e.g., large-scale network optimization with acceptable delays.
SA Greedy 3-opt	Combines SA with 3-opt moves, enhancing solution accuracy and convergence but at the cost of increased runtime; parameter tuning is crucial.	Scales better than exact methods but requires careful balancing of runtime and quality, especially for larger graphs.	Suitable for applications demanding higher solution quality without strict optimality, such as complex vehicle routing with preference for accurate solutions.
GA Tournament Selection	Evolving solutions via tournament selection achieves reasonable accuracy but can be computationally expensive; solution quality depends on population size and mutation rate.	Scales moderately well but runtime increases with population size and generations, limiting feasibility for very large instances (N > 50).	Useful in combinatorial optimization tasks with multiple constraints, such as resource allocation or workforce scheduling in dynamic environments.
GA Adaptive Mutation	Adaptive mutation improves convergence speed and solution quality but requires more computational resources; balancing exploration and exploitation is challenging for large graphs.	Scales moderately, with adaptive mutation aiding convergence but limited feasibility for very large graphs.	Suitable for evolving solutions over time in changing conditions, such as adaptive scheduling in manufacturing or load balancing in network management.
LK	Provides near-optimal solutions for small to medium instances, but becomes computationally intensive on larger graphs (N > 200). It's a heuristic, so it doesn't guarantee optimality but often achieves close-to-optimal results.	Scales well for medium-sized graphs but becomes slower on large instances due to complex move evaluations.	Effective in routing and planning scenarios where near-optimal solutions are needed, especially for medium-scale problems like last-mile delivery or network optimization with moderate node counts.

Future Work

Future research could focus on **parallelization** of computationally intensive algorithms like Held-Karp, Lin-Kernighan, and Genetic Algorithms (GA) to improve scalability for larger instances. **Hybrid approaches**, such as combining Simulated Annealing (SA) or GA with Lin-Kernighan (LK) for refinement, may also offer a balance between accuracy and speed. Exploring **adaptive parameter tuning** in SA and GA could enhance their performance by dynamically adjusting parameters during execution. Additionally, **machine learning-based heuristics**, such as reinforcement learning, could be applied to reduce the search space, accelerating convergence. Finally, benchmarking on **real-world datasets** with practical constraints would provide insights into the applicability of these algorithms in domains like logistics and supply chain optimization.

Conclusion

This report provides a comparative analysis of TSP algorithms based on solution cost, execution time, and accuracy. Held-Karp offers exact solutions but is limited to very small instances, while Greedy and Christofides scale well, with the latter theoretically providing a 1.5-approximation for symmetric TSPs, although most TSPLIB datasets used in this analysis do not provide the necessary conditions for optimality. Simulated Annealing (SA) and Genetic Algorithms (GA) allow for flexible accuracy and are effective on larger instances but require careful tuning. Lin-Kernighan (LK) provides high-quality solutions for medium-sized problems, balancing accuracy and speed.

SA emerged as effective options for large instances, balancing runtime and solution quality, while for **Christofides**, more analysis will need to be done on more “ideal” datasets to confirm its viability, performance-wise. In contrast, **LK and Memetic GA** excel in medium-sized cases, often achieving near-optimal solutions. For smaller-scale problems, **Held-Karp** is the definitive solution, providing exactly optimal solutions. Future work could explore parallelization, hybrid methods, adaptive tuning, and machine learning to enhance scalability and applicability in real-world scenarios. These insights underscore the importance of selecting algorithms based on problem size, accuracy needs, and computational resources.