Various Travelling Salesman Problems

香 港 大 學
THE UNIVERSITY OF HONG KONG

Liu Zhonglin

MATH3999: Directed Studies in Mathematics

Supervised by: Prof. Zang Wenan

2024-2025 Semester 1

Abstract

The Traveling Salesman Problem (TSP) remains a quintessential NP-hard challenge in combinatorial optimization. This report presents a comparative study evaluating the performance of classical approximation algorithms against modern reinforcement learning (RL) approaches for solving the Euclidean TSP. We implement and analyze two established heuristics, the Quick Insertion method and Christofides' algorithm, alongside a Pointer Network-based RL agent. To investigate the impact of learning strategy, the RL agent is trained under three distinct paradigms: a pure policy gradient (REINFORCE) approach, a hybrid REINFORCE model initialized with a Christofides-ordered input sequence, and a more advanced hybrid Actor-Critic model using the same structured input. A robust evaluation is conducted on randomly generated 30-city and 100-city problem instances. The results demonstrate that while classical heuristics deliver superior solution quality and computational speed, providing the RL agent with a structural "hint" from a classical algorithm dramatically improves its performance. Most notably, the hybrid model trained with an Actor-Critic algorithm successfully scales to the larger problem size, achieving an average tour length competitive with the classical heuristics. This study concludes that the potential of learning-based solvers for the TSP is significantly unlocked through the synergistic integration of classical algorithmic structure with stable, advanced reinforcement learning techniques.

1 Introduction

The Traveling Salesman Problem (TSP) is a quintessential challenge in combinatorial optimization that has been studied since at least 1930 [1]. This problem involves finding the shortest possible route that visits each city in a list exactly once and returns to the origin city, encapsulating a broad range of applications from logistics to DNA sequencing [2].

Historically, Merrill M. Flood was among the first to formalize the problem during his exploration of school bus routing problems, marking the beginning of mathematical approaches to TSP [1]. The TSP is formulated on an undirected graph where cities are vertices and paths are edges with associated costs, making it a model for numerous theoretical and practical problems.

Despite its NP-hard status, indicating that no efficient algorithm is known to exist for solving all cases of TSP efficiently, researchers have developed a variety of heuristic and exact algorithms. These methods can handle problems with millions of cities, often producing solutions close to optimal [3]. The TSP has also inspired numerous extensions and variations, addressing more complex or specialized scenarios such as the time windows problem [4], the multi-objective problem [5], and even more exotic forms like the bottleneck TSP [6].

In parallel to these classical developments, the rise of deep learning has introduced a new paradigm for tackling such problems. Reinforcement Learning (RL), in particular, offers the potential to learn effective solution policies directly from experience, providing a flexible alternative that may excel in complex environments where traditional heuristics fall short. This study bridges the gap between these two domains by implementing and rigorously comparing two classical heuristics against three paradigms of a deep reinforcement learning agent, aiming to quantify the performance trade-offs and explore the efficacy of hybrid methodologies.

2 Preliminaries and Problem Formulation

Before delving into the problem formulation of the TSP, it is essential to introduce several foundational concepts that play a critical role in understanding and solving the problem. Let $G = (V, E)$ be a graph where V is the set of vertices and E is the set of edges.

Definition 1 (Matching). In a graph $G = (V, E)$, a *matching* is a subset of edges $M \subseteq E$ such that no two edges share a common vertex. A matching M is called perfect matching if $|M| = \frac{|V|}{2}$, i.e. each vertex of G is incident with an edge in M.

Theorem 1. Let G be a graph with a weight $w(e)$ on each edge. Then the maximum-weight matching, maximum-weight perfect matching, minimum-weight matching, and

minimum-weight perfect matching problems can be solved in polynomial time.

Definition 2 (Minimum Spanning Tree). In a connected, edge-weighted graph $G = (V, E)$, a *Minimum Spanning Tree (MST)* is a subset of the edges $T \subseteq E$ that connects all the vertices together without any cycles and with the minimum possible total edge weight.

Definition 3 (Hamiltonian Cycle). In a graph $G = (V, E)$, a *Hamiltonian cycle* is a cycle that visits each vertex exactly once and returns to the starting vertex.

Definition 4 (Eulerian Tour). In a graph $G = (V, E)$, an *Eulerian tour* is a tour that visits every edge exactly once and returns to the starting vertex.

Definition 5 (Eulerian Graph). A graph $G = (V, E)$ is called an *Eulerian graph* if there exists an Eulerian tour in the graph.

Theorem 2 (Characterization of Eulerian Graphs). A connected graph $G = (V, E)$ is Eulerian if and only if every vertex of the graph has an even degree.

We will now introduce classical algorithms for finding spanning trees and Eulerian tours in Eulerian graphs.
Kruskal's algorithm and Prim's algorithm process a connected graph $G = (V, E)$ with a weight $w(e)$ on each edge e, and output the minimum spanning tree of the graph.

Algorithm 1 Kruskal's Algorithm

```
F ← ∅
for v ∈ V do
    MAKE-SET(v)
end for
Sort all edges e ∈ E by weight w(e), in non-decreasing order
for each edge {u, v} in the sorted list do
    if FIND-SET(u) ≠ FIND-SET(v) then
        F ← F ∪ {{u, v}}
        UNION(FIND-SET(u), FIND-SET(v))
    end if
end for
return F
```

Algorithm 2 Prim's Algorithm

1: Find the edge $e = \{u, v\}$ in E with the minimum weight $w(e)$
2: Initialize the tree T with the edge e, $T := \{e\}$
3: Initialize the set of vertices in T, $V_T := \{u, v\}$
4: **while** $|V_T| < |V|$ **do**
5: Find the edge $\{x, y\}$ in E with the minimum weight $w(\{x, y\})$, where $x \in V_T$ and $y \notin V_T$
6: Add $\{x, y\}$ to T
7: Add y to V_T
8: **end while**
9: **return** T

Fleury's algorithm is designed to find an Eulerian tour in a graph.

Algorithm 3 Fleury's Algorithm

1: Choose an arbitrary vertex v_0.
2: Initialize $tour = [v_0]$.
3: **while** there are unused edges in the graph **do**
4: Let v be the current vertex at the end of the last added edge in *tour* (start with v_0).
5: Choose e from the edges connected to v such that e is not a bridge, unless no other option exists.
6: Add e and the vertex at the other end of e to *tour*.
7: Remove e from the graph.
8: **end while**
9: **return** *tour*

Travelling salesman problem:
A salesman wishes to make a tour, visiting each of n cities exactly once and finishing at tyhe city he starts from. Suppose there is a road between any two cities and the road between city i and city j is of distance w_{ij}. The objective of the salesman is to find such a tour with the minimum total distance.
This problem can be formulated in the following form:

Travelling salesman problem(Equivalent Form):
Input: A complete graph $G = (V, E)$ with a weight w_{ij} on each edge ij.
Output: A Hamiltonian cycle of G with minimum total weight.

3 Classical Approximation Algorithms

Since the TSP is NP-hard, there is no polynomial-time algorithm for solving it unless $NP = P$. So scientists turn to search for approximation algorithm.
A polynomial-time algorithm is said to be a **δ-approximation algorithm** for TSP if for

every instance I, it delivers a Hamiltonian cycle C such that

$$\frac{w(C)}{w(C^\star)} \leq \delta,$$

where $C^\star$ is a Hamiltonian cycle of I with minimum total weight.
A weight function w defined on a complete graph G is said to satisfy the **triangle-inequality** if for any three vertices i, j, k of G,

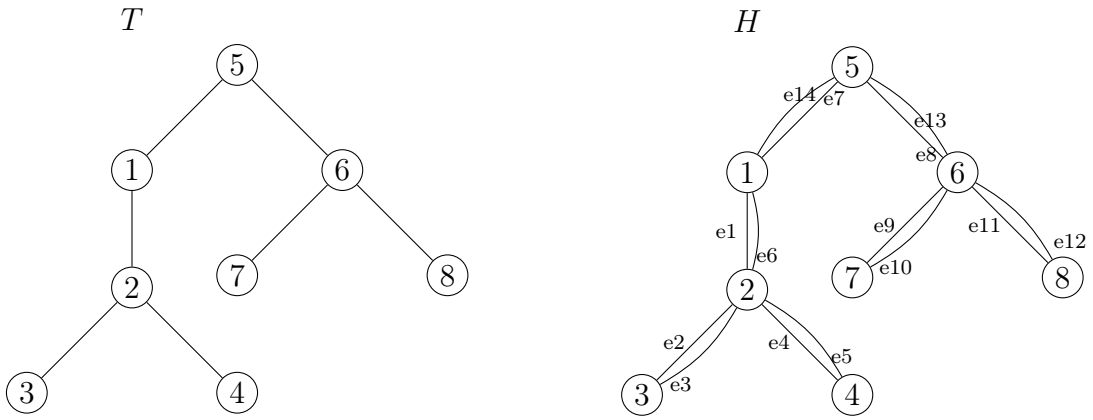
$$w_{ij} + w_{jk} \geq w_{ik}.$$

In this project we shall investigate some approximation algorithms for the TSP satisfying triangle-inequalities. Let $G = (V, E)$ be a complete graph with a nonnegative integral weight on each edge, and the weight function satisfies triangle-inequality.

3.1 Quick Insertion Method

Algorithm 4 Quick Insertion Method

1: Compute a minimum spanning tree T in G.
2: Let H be the multigraph obtained from T be duplicating each edge once. Compute an Eulerian tour W of H, and connect W into a Hamiltanian cycle C by making a shortcut. Return C.



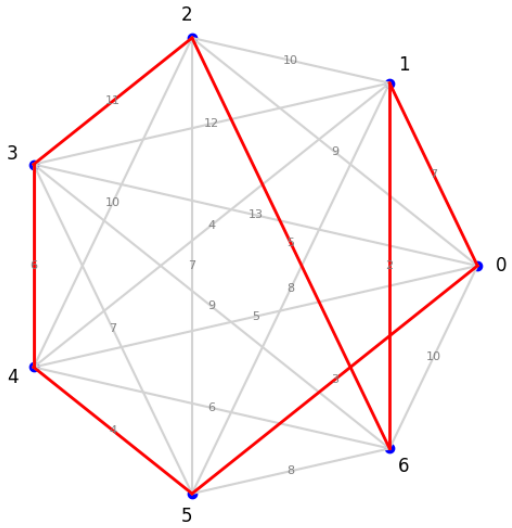
Remark 1. Recall that W is a sequence where terms are alternately vertices and edges. The short cut can be produced by listing the vertices when they're first encountered in W. Suppose T is as depicted above. Then an Eulerian tour on H is

$$W = 1e_1 2e_2 3e_3 2e_4 4e_5 2e_6 1e_7 5e_8 6e_9 7e_{10} 6e_{11} 8e_{12} 6e_{13} 5e_{14} 1.$$

Then $C = 123456781$ is the desired Hamiltonian cycle. Since the weight function satisfies triangle inequality, $w_{34} \leq w(e_3) + w(e_4), w_{45} \leq w(e_5) + w(e_6) + w(e_7), w_{78} \leq w(e_{10}) +$

$w(e_{11}), w_{81} \leq w(e_{12}) + w(e_{13}) + w(e_{14})$. Thus $w(C) \leq w(W) \leq 2w(C^\star)$.

Below is a graph illustrating the implementation of the algorithm. The resulting tour is $[0, 1, 6, 2, 3, 4, 5, 0]$.



Theorem 3. The performance guarantee of Quick Insertion Method is 2.

3.2 Christofides Algorithm

Algorithm 5 Christofides' Algorithm (1976)

1: Compute a minimum spanning tree T in G.
2: Let U denote the set of all vertices with odd degree in T. Compute a minimum weighted perfect matching M in $G[U]$, the graph induced by U in G.
3: Compute an Eulerian tour W on $H = T + M$. Convert W into a Hamiltonian cycle C by making a short cut. Return C.

We implement our code here, which demonstrates the algorithm running on a complete graph with 5 vertices:

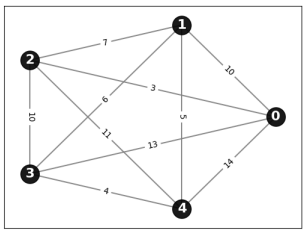


Figure 1: Initial Placement

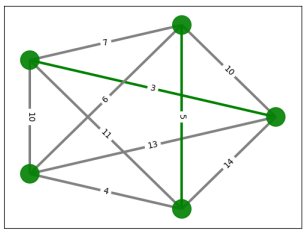


Figure 2: Finding MST

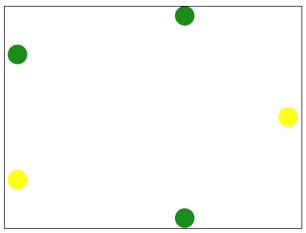


Figure 3: Vertices of odd degree in MST

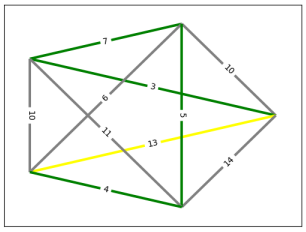


Figure 4: Minimum weight matching in induced subgraph

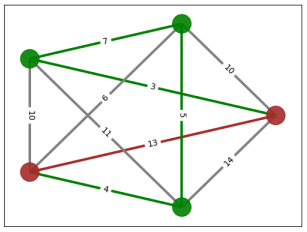


Figure 5: Combine the matching and MST

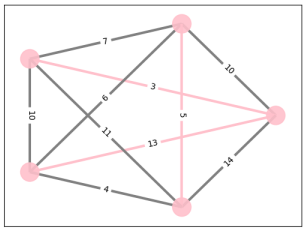


Figure 6: Eulerian circuit

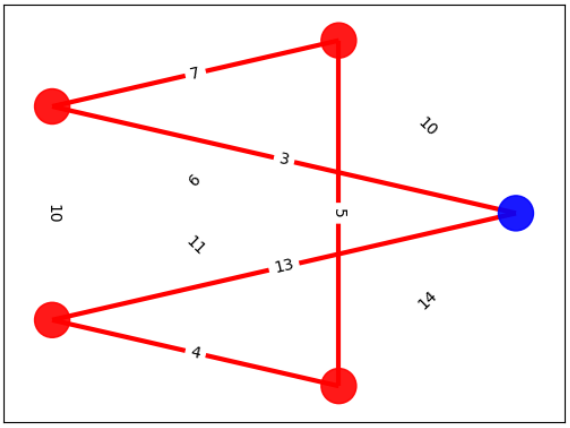


Figure 7: Final Hamiltonian cycle

Theorem 4. The performance guarantee of Christofides algorithm is 1.5.

Proof. Since the weight function satisfies the triangle inequality, according to the construction of C, we have

$$c(C) \leq c(W) = c(T) + c(M), \tag{1}$$

where $c(A)$ stands for the total weight of A with respect to c.
Now let $C^\star$ be a minimum Hamiltonian cycle of G. Then

$$c(C^\star) \geq c(T). \tag{2}$$

Indeed, for any edge e on $C^\star, C^\star - e$ is a spanning tree of G. So $c(C^\star - e) \geq c(T)$ and thus 2 follows.
Let $\{i_1, i_2, \ldots, i_2 m\}$ be the vertices in U in step 2, in the order they appear in $C^\star$. Consider

two matchings of U:

$$M_1 = \{i_1 i_2, i_3 i_4, \ldots, i_{2m-1} i_{2m}\},$$
$$M_2 = \{i_2 i_3, i_4 i_5, \ldots, i_{2m} i_1\}.$$

By the triangle inequality, $c(C^\star) \geq c(M_1) + c(M_2)$. Since M is a minimum perfect matching of $G[U], C(M_1) \geq C(M)$ and $c(M_2) \geq C(M)$. Thus $c(C^\star) \geq 2c(M)$ or

$$c(M) \leq \frac{c(C^\star)}{2} \tag{3}$$

Combining 1, 2 and 3, we have

$$c(C) \leq 1.5c(C^\star).$$

□

Additionally, an animated demonstration of the Christofides algorithm was developed using the Manim animation engine, available for viewing at: `https://liu-zhonglin.github.io/Christofides-vs-RL-for-TSP/Project%20Report/Christofides.mp4`

4 Reinforcement Learning for the TSP

4.1 Model Architecture: Pointer Network with Attention

The core of the learning-based approach is a sequence-to-sequence model known as a Pointer Network [7]. This architecture is particularly well-suited for combinatorial optimization problems where the output is a permutation of the input, a task for which standard sequence-to-sequence models that output to a fixed-size dictionary are ill-suited. The model leverages an encoder-decoder structure augmented with an attention mechanism, a paradigm that has proven highly effective in fields such as natural language translation bahdanau2014neural.

The model's process begins with an encoder, implemented as a Long Short-Term Memory (LSTM) network. The encoder's role is to create a rich, contextual representation of the entire input TSP problem. Given an input sequence of n city coordinates $X = (x_1, x_2, \ldots, x_n)$, where each $x_i \in \mathbb{R}^2$, the encoder processes this sequence to produce a set of hidden state embeddings $E = (e_1, e_2, \ldots, e_n)$. Each embedding $e_j \in \mathbb{R}^h$ (for a hidden dimension h) contains information about a specific city in the context of all other cities. The final hidden and cell states of the encoder, (h_n, c_n), serve as a summary of the entire problem instance and are used to initialize the decoder.

The decoder, also an LSTM, is tasked with autoregressively constructing the output tour $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ by "pointing" to one of the input cities at each decoding step.

This pointing mechanism is achieved through a content-based attention mechanism. At each decoding step i, the decoder's current hidden state, d_i, is used as a "query" to the attention mechanism. This mechanism computes an alignment score, u_{ij}, for each of the encoder's hidden states e_j, measuring how relevant input city j is to the current state of the tour construction. We use additive attention, formulated as:

$$u_{ij} = v^T \tanh(W_1 e_j + W_2 d_i)$$

where W_1, W_2, and v are learnable weight parameters.

To ensure a valid TSP tour is generated, a mask is applied to the scores at each step, setting the score of any previously visited city to $-\infty$. This prevents the model from selecting the same city twice. The masked scores are then converted into a probability distribution over the input cities using the softmax function, which represents the model's policy:

$$p(\pi_i = j | \pi_{1..i-1}, X) = \frac{\exp(u_{ij})}{\sum_{k=1}^{n} \exp(u_{ik})}$$

The city with the highest probability is then selected as the next city in the tour, π_i, and its coordinates become the input for the subsequent decoder step. This process repeats for n steps, generating a full permutation of the input cities, which constitutes the final TSP tour.

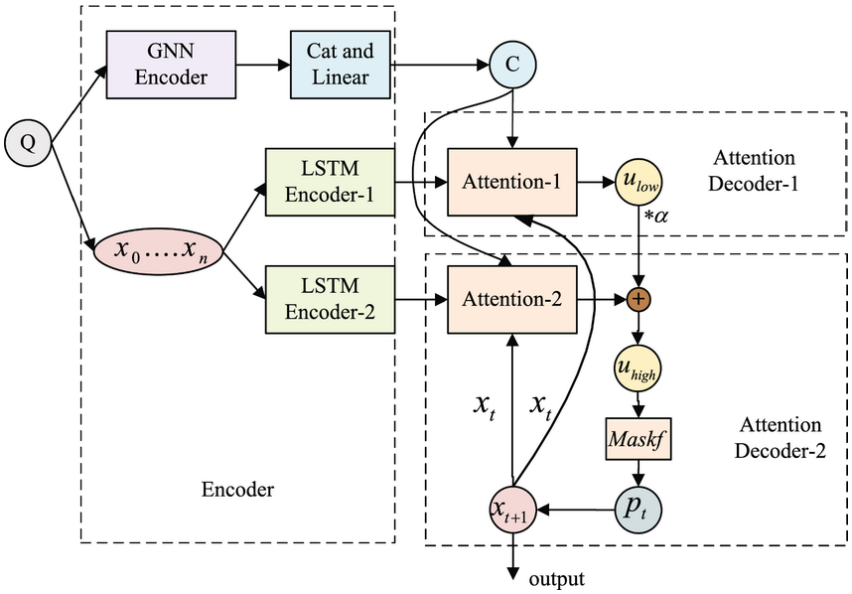


Figure 8: A simplified diagram of the Pointer Network architecture, showing the encoder processing all inputs and the decoder pointing to one input at each step via the attention mechanism.

4.2 Training Paradigms

To thoroughly investigate the impact of different learning strategies, three distinct RL agents were trained on 30-city problems, each for 5000 epochs. The best-performing model weights from each training run were preserved for the final evaluation.

4.2.1 Pure RL with REINFORCE

The first model served as a baseline to evaluate the performance of the Pointer Network architecture in isolation. It was trained using the REINFORCE policy gradient algorithm [8]. For each training step, a batch of TSP instances was generated with city coordinates in a random, unordered sequence. The model's objective is to minimize the expected tour length. The loss function for a single instance is defined as:

$$\mathcal{L}(\theta) = \mathbb{E}_{\pi_\theta}[L(\pi)]$$

where $L(\pi)$ is the length of the tour π generated by the policy π_θ with parameters θ. The gradient is estimated using the REINFORCE rule:

$$\nabla_\theta \mathcal{L}(\theta) \approx L(\pi) \nabla_\theta \log p(\pi|X;\theta)$$

Here, $L(\pi)$ serves as the reward signal. By performing gradient descent on this objective, the model updates its weights to increase the probability of generating shorter tours.

4.2.2 Hybrid RL with REINFORCE

The second agent was designed to test the hypothesis that providing structural information from a classical algorithm could improve learning. The training process was identical to the pure RL model, using the same REINFORCE algorithm. However, the input sequence of city coordinates, X, was not random. Instead, for each problem instance, the Christofides algorithm was first run to generate an Eulerian tour. This tour was then converted into a valid Hamiltonian cycle via a deterministic shortcutting heuristic. The resulting high-quality tour provided the ordered sequence of cities that was fed into the Pointer Network. The agent's task was thus transformed from solving the problem from scratch to learning how to improve upon an already excellent solution.

4.2.3 Hybrid RL with Actor-Critic

The third and most advanced agent builds upon the hybrid approach by using a more stable training algorithm known as an Actor-Critic method [9]. This paradigm introduces a second neural network, the Critic (V_ϕ), whose purpose is to estimate the value (expected tour length) of a given state, thereby providing a more refined learning signal.

The training process begins with the Actor (π_θ, our Pointer Network) taking the Christofides-ordered city sequence and producing a tour π along with its log-probability, $\log p(\pi|X;\theta)$. The final hidden state from the Actor's decoder, s, is then passed to the Critic, which outputs a value estimate, $V_\phi(s)$. This value represents the Critic's prediction of the tour length given the state. The actual tour length, $L(\pi)$, is then calculated to serve as the ground-truth reward.

A key concept in this method is the Advantage, A, which measures how much better or worse the Actor's tour was compared to the Critic's expectation. It is defined as $A = L(\pi) - V_\phi(s)$. To prevent the Actor's loss from influencing the Critic's training, the value term $V_\phi(s)$ is detached from the computation graph during this step. The model is then updated by optimizing a combined loss function. The Actor Loss is designed to reinforce actions that led to a positive advantage, effectively encouraging policies that outperform the Critic's baseline. Its gradient is given by:

$$\nabla_\theta \mathcal{L}_{actor}(\theta) = -A \cdot \nabla_\theta \log p(\pi|X;\theta)$$

Simultaneously, the Critic Loss is updated via a standard Mean Squared Error loss, which trains the Critic to make its value predictions, $V_\phi(s)$, more accurate with respect to the true tour length, $L(\pi)$:

$$\mathcal{L}_{critic}(\phi) = (L(\pi) - V_\phi(s))^2$$

This method generally leads to more stable training than REINFORCE because the advantage signal is less noisy than using the raw reward alone.

5 Results and Discussion

To provide a definitive and statistically sound comparison, a final robust evaluation was conducted. Each of the five developed methods was tested on 1000 unique, randomly generated TSP instances for both 30-city and 100-city problem sizes. The average tour length and average computation time were recorded to compare performance and scalability.

The TSP instances were generated by uniformly sampling city coordinates in a $[0, 1] \times [0, 1]$ unit square. This ensures that the edge weights, calculated as the Euclidean distance between cities, satisfy the triangle inequality, making it a valid Metric TSP for which the classical heuristics are appropriate.

The key hyperparameters for all reinforcement learning models were kept consistent to ensure a fair comparison. A hidden dimension of 128 was used for the LSTM cells in the Pointer Network, a standard size that provides sufficient capacity for learning complex patterns. A batch size of 128 was used during training to offer a good balance between stable gradient estimates and memory usage. For the Adam optimizer, a learning rate of 1×10^{-4} was chosen, as a small rate is standard for policy gradient methods to prevent

destructively large updates and promote stable convergence. All RL models were trained for 5000 epochs to provide ample opportunity to converge, and the model weights that achieved the best average tour length during this period were saved and used for the final evaluation. The final evaluation itself was performed over 1000 trials for each problem size to ensure the reported average performance is statistically robust and not the result of outlier instances.

5.1 Experimental Results

The aggregated results from the 1000-trial evaluations for both problem scales are presented below in Table 1. Figure 9 provides a visual summary of the tour length results, illustrating how each algorithm's performance scales with problem size.

Table 1: Final Robust Average Results (1000 Trials)

Metric		Christofides	Nearest Insertion	Pure RL	Hybrid (REINFORCE)	Hybrid (Actor-Critic)
Avg. Tour Length	30 Cities	5.3300	5.2107	11.9242	10.5369	10.3699
	100 Cities	9.3701	9.3125	32.4112	26.9063	9.3705
Avg. Comp. Time (s)	30 Cities	0.0005	0.0005	0.0222	0.0218	0.0216
	100 Cities	0.0013	0.0137	0.0740	0.0733	0.0741

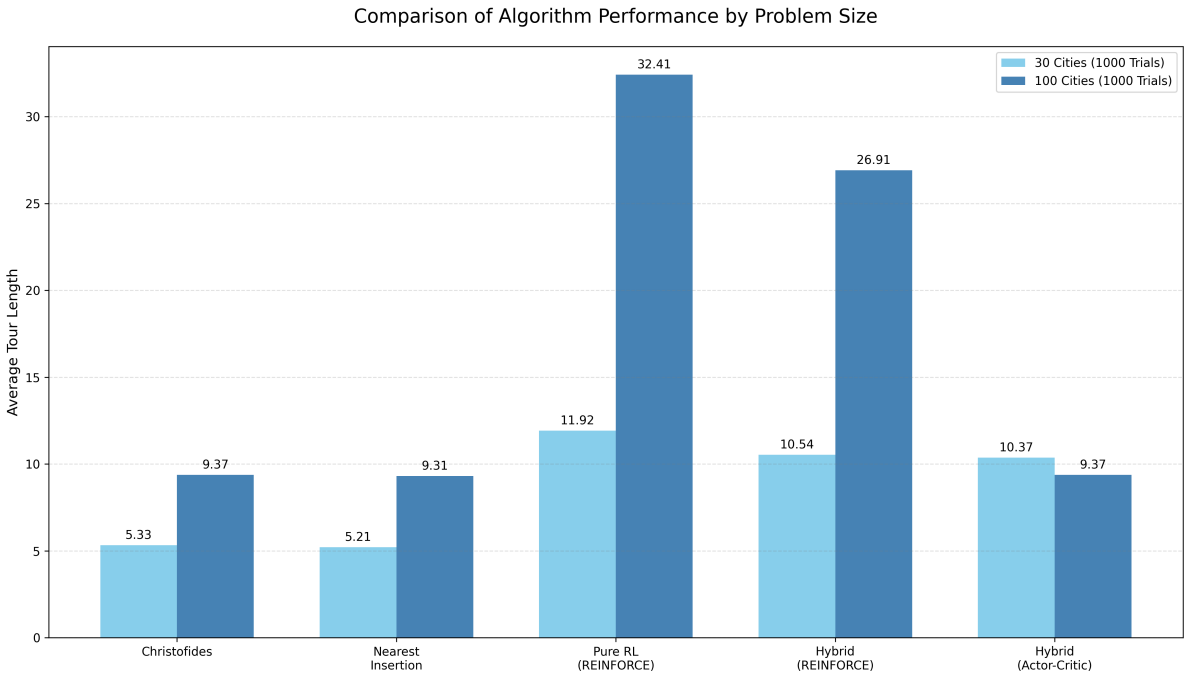


Figure 9: A grouped bar chart comparing the average tour length of each algorithm across both 30-city and 100-city problem sizes.

5.2 Discussion

The comprehensive evaluation reveals several key insights into the performance and characteristics of these different approaches.

Across both 30-city and 100-city scales, the classical heuristics proved vastly superior to the reinforcement learning models in both solution quality and computational speed. Interestingly, the simpler Nearest Insertion heuristic consistently found slightly better average solutions than the Christofides algorithm, which has a stronger theoretical worst-case guarantee. This highlights a common phenomenon in practice where simpler, greedy methods can outperform more complex algorithms on average problem instances. Their sub-second computation times confirm their suitability for real-world applications where speed is critical.

A crucial finding is the dramatic performance gap between the pure RL model and the hybrid models. At both scales, the pure RL agent, which learned from randomly ordered inputs, produced significantly longer tours than the hybrid agents, which were given a Christofides-ordered sequence. This demonstrates that providing the learning agent with a high-quality structural "hint" from a classical algorithm is a highly effective strategy for improving performance and guiding the learning process.

The most striking result is the performance of the Hybrid (Actor-Critic) model on the 100-city problem. While the simpler Hybrid (REINFORCE) model's performance degraded significantly at the larger scale, the Actor-Critic model's performance was outstanding, achieving an average tour length statistically identical to that of the powerful Christofides and Nearest Insertion heuristics. This shows that for complex, large-scale problems, a stable learning algorithm like Actor-Critic is not just a minor improvement; it is the critical factor that enables the RL agent to effectively learn a competitive policy.

This study confirms that while classical heuristics remain the benchmark for the standard Euclidean TSP, the potential of reinforcement learning is unlocked through intelligent integration with classical methods. The success of the Actor-Critic hybrid model demonstrates that by combining the structural knowledge of classical algorithms with the stable learning signal of advanced RL techniques, it is possible to train agents that can achieve performance competitive with strong, specialized heuristics.

6 Conclusion

This directed study successfully implemented and rigorously evaluated a range of solution methods for the Traveling Salesman Problem, from established classical heuristics to modern deep reinforcement learning agents. The comprehensive, multi-trial evaluation across different problem scales provided a clear and definitive performance hierarchy.

The results reaffirm the efficacy and efficiency of classical approximation algorithms

like Nearest Insertion and Christofides, which consistently produced the highest-quality solutions in a fraction of the time required by the learning-based models. However, the study's most significant contribution lies in its investigation of hybrid reinforcement learning. The experiments conclusively demonstrate that while a pure RL agent struggles to scale effectively, its performance can be dramatically improved by integrating domain knowledge from classical algorithms. Furthermore, the outstanding performance of the Actor-Critic hybrid model on the 100-city problem underscores the critical importance of using advanced, stable training algorithms to tackle complex optimization tasks.

Ultimately, this research suggests that the most promising path forward in applying machine learning to combinatorial optimization is not to replace classical methods, but to create sophisticated hybrid systems. By leveraging the structural insights of classical theory to guide the powerful, flexible learning capabilities of modern neural networks, it is possible to develop agents that can achieve performance competitive with strong, specialized heuristics.

7 Code Availability

The complete Python source code for all classical algorithms, reinforcement learning models, training scripts, and evaluation protocols developed in this study is publicly available on GitHub at the following repository: `https://github.com/Liu-Zhonglin/Christofides-vs-RL-for-TSP`

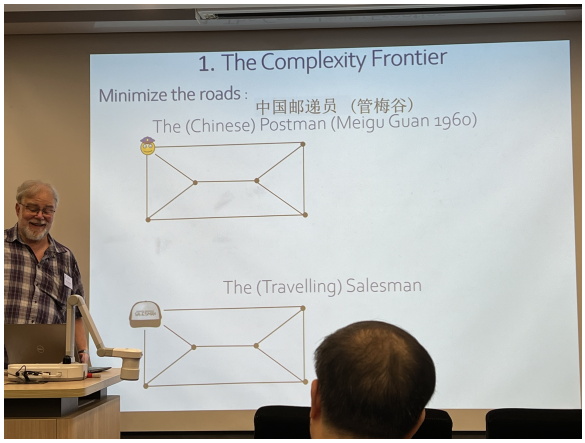
8 Acknowledgment

I would like to extend my sincere gratitude to Prof. Zang Wenan, a leading expert in graph theory and discrete optimization, for his invaluable guidance and support throughout my capstone course, MATH3999: Directed Studies in Mathematics. His insightful advice has been instrumental in shaping my understanding of graph theory and discrete optimization, as well as in directing my independent research. Under his recommendation, I concurrently enrolled in MATH3943: Network Models in Operations Research during the academic year 2024. This course provided an excellent introduction to graph theory and a solid foundation for exploring the theoretical aspects of algorithms, complementing the research I undertook in MATH3999.

I would also like to express my heartfelt appreciation to the Mathematics Department of HKU, which fosters creative thinking and provides a strong foundation for both teaching and research. I was honored to attend the Frontiers of Mathematics Lecture, themed "The Salesman and the Postman: Frontiers and Gateways in Combinatorial Optimization", organized by the department. This lecture offered valuable insights into how

the Chinese Postman Problem can inspire innovative approaches to tackling the Traveling Salesman Problem (TSP), further enriching my understanding of combinatorial optimization.

(a) Frontiers of Mathematics Lecture (1)



(b) Frontiers of Mathematics Lecture (2)

Figure 10: Photos from the Frontiers of Mathematics Lecture.

In conclusion, I am deeply grateful for the opportunities and support I have received during this capstone course and my studies at HKU. These experiences have significantly enhanced my knowledge and skills in mathematics, particularly in the field of optimization. I look forward to applying what I have learned to future challenges and contributing to advancements in this exciting area of research

References

[1] M Flood and LJ Savage. A game theoretic study of the tactics of area defense. *RAND Research Memorandum*, 51, 1948.

[2] Rajesh Matai, Surya Prakash Singh, and Murari Lal Mittal. Traveling salesman problem: an overview of applications, formulations, and solution approaches. *Traveling salesman problem, theory and applications*, 1(1):1–25, 2010.

[3] D.L. Applegate, R.E. Bixby, V. Chvátal, and W.J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton Series in Applied Mathematics. Princeton University Press, 2011. ISBN 9781400841103. URL `https://books.google.com.hk/books?id=zfIm94nNqPoC`.

[4] Gilbert Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(3):345–358, 1992. ISSN 0377-2217. doi: https://doi.org/10.1016/0377-2217(92)90192-C. URL `https://www.sciencedirect.com/science/article/pii/037722179290192C`.

[5] Iraklis-Dimitrios Psychas, Eleni Delimpasi, and Yannis Marinakis. Hybrid evolutionary algorithms for the multiobjective traveling salesman problem. *Expert Systems with Applications*, 42(22):8956–8970, 2015. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2015.07.051. URL https://www.sciencedirect.com/science/article/pii/S0957417415005114.

[6] R. S. Garfinkel and K. C. Gilbert. The bottleneck traveling salesman problem: Algorithms and probabilistic analysis. *J. ACM*, 25(3):435–448, July 1978. ISSN 0004-5411. doi: 10.1145/322077.322086. URL https://doi.org/10.1145/322077.322086.

[7] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper_files/paper/2015/file/29921001f2f04bd3baee84a12e98098f-Paper.pdf.

[8] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

[9] Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.