Q-Learning Guided Local Search for the Traveling Salesman Problem

Sanaa El Jaghaoui*, Aissa Kerkour Elmiad
Laboratory LARI-Faculty of Sciences,
Mohammed First University, Oujda, Morocco

***Abstract*—The Traveling Salesman Problem (TSP) remains a fundamental challenge in combinatorial optimization with applications in logistics, routing, and network design. Classical local search methods face a trade-off between solution quality and computational efficiency: while 3-opt delivers better solutions than 2-opt, its $O(n^3)$ complexity renders it impractical for large instances. This paper presents a reinforcement learning (RL) approach that addresses this challenge through intelligent guidance of local search operators. Our method employs a simple one-dimensional Q-table that learns to identify poorly positioned cities and directs 2-opt and 3-opt operations toward the most promising tour segments. We evaluate the approach on 55 TSPLIB benchmark instances ranging from 51 to 18,512 cities. For instances up to 1,000 cities, RL-guided 3-opt (RL-3opt) achieves optimality gaps of 0.9–2.2% compared to 3.8–4.3% for classical 3-opt, with execution times reduced from hours to under one second and speedups reaching 32,323×. For instances between 1,000–5,000 cities, RL-3opt maintains computational efficiency (100–30,000× speedups) while achieving competitive 6.3% gaps. Both RL-2opt and RL-3opt execute in sub-second to a few seconds even on problems with over 18,000 cities. All experiments run on standard CPU hardware without GPU acceleration, demonstrating that effective TSP optimization remains accessible without specialized resources.**

***Keywords*—Traveling salesman problem; reinforcement learning; Q-Learning; local search; 2-opt; 3-opt**

I. Introduction

The Traveling Salesman Problem (TSP) asks for the shortest route visiting each city exactly once before returning to the origin [1]. Given n cities with distances d_{ij} between pairs, the objective is to find a permutation $\pi = \{s_1, s_2, \ldots, s_n\}$ minimizing:

$$\sum_{k=1}^{n-1} d_{s_k s_{k+1}} + d_{s_n s_1}. \quad (1)$$

Despite this simple formulation, the TSP is NP-hard [2], with $(n-1)!/2$ possible tours for symmetric instances where $d_{ij} = d_{ji}$. This computational difficulty has motivated decades of research, with applications spanning logistics, manufacturing, circuit design, and delivery routing [3]. TSP solution methods have evolved through several generations. Classical heuristics combine construction methods with local search operators and remain widely used for their simplicity and reliability. Metaheuristic algorithms offer more sophisticated search strategies that balance exploration and exploitation through mechanisms such as population evolution, adaptive search, and strategic diversification. More recently, machine learning techniques, particularly deep neural networks and RL, have shown they can learn solution strategies directly from data [4], [5]. Researchers have also developed hybrid methods that combine classical algorithms with learning components [6], [7].

Despite these advances, two fundamental challenges persist. Most learning-based approaches require substantial computational resources [8], [9]. Deep learning methods typically need GPU acceleration and extensive training on large datasets [10]. Even simpler learning frameworks often employ multi-dimensional state representations that scale poorly with problem size. This limits their accessibility for users without high-performance computing infrastructure, including small businesses, educational institutions, and organizations in developing regions. At the same time, classical local search methods face an inherent trade-off between speed and solution quality. The 2-opt operator runs quickly but often settles at poor local minima because it only examines a limited neighborhood around the current solution. The 3-opt operator explores substantially larger neighborhoods and typically finds better solutions, but at much higher computational cost. On large instances, 3-opt can require hours or even days to complete. This creates a practical dilemma: users must choose between fast but suboptimal solutions, or high-quality solutions that may be computationally prohibitive.

We address both challenges through a simple RL framework designed for efficiency. Our method uses a one-dimensional Q-table indexed by cities rather than state-action pairs, reducing memory complexity from $O(n^2)$ to $O(n)$. During tour improvement, the table learns which cities are poorly positioned based on their involvement in successful optimization moves. These learned priorities then guide both 2-opt and 3-opt operators to focus on tour segments most likely to benefit from modification. This design combines the simplicity and accessibility of classical methods with the adaptive intelligence of learning-based approaches. Three principles shape our work: prioritize algorithmic simplicity, learn during optimization rather than through offline training, and execute on standard CPU hardware without GPU acceleration.

We validate the approach on 55 TSPLIB benchmark instances [11] ranging from 51 to 18,512 cities. To directly measure the contribution of RL, we include systematic comparisons against classical baselines: nearest neighbor construction, classical 2-opt, and classical 3-opt. Results demonstrate that RL-guided methods outperform their classical counterparts in solution quality for instances up to 1,000 cities. For larger instances (1,000-5,000 cities), RL-3opt achieves competitive quality (6.3% gap) compared to classical 3-opt (5.5% gap) while running 100 to over 30,000 times faster, effectively

*Corresponding author.

transforming hours or days into seconds on large instances. Both RL-2opt and RL-3opt complete in a few seconds even on problems with over 18,000 cities, where classical 3-opt becomes entirely impractical. All experiments run on standard CPU hardware, confirming that the approach requires no specialized computing resources.

This research aims to: 1) develop an efficient RL-guided framework reducing $O(n^3)$ complexity while maintaining solution quality, 2) ensure accessibility through a minimalist one-dimensional Q-table operating on standard CPU hardware, 3) demonstrate scalability across 51 to 18,512-city instances, 4) achieve sub-second to few-second execution times for real-time applications, and 5) quantify quality-time trade-offs compared to classical methods.

The paper proceeds as follows. Section II reviews related work on TSP solution methods and learning-based optimization. Section III describes our approach in detail, including RL fundamentals, classical local search methods, and our RL-guided framework. Section IV presents experimental methodology and comprehensive results on 55 TSPLIB benchmarks. Section V concludes the paper.

II. Related Work

The TSP has been extensively studied through classical heuristics, metaheuristics, and modern learning-based approaches. This section reviews key methodologies relevant to our hybrid framework.

A. Classical Heuristics and Local Search

Classical heuristics provide rapid initial solutions. The nearest neighbor heuristic [12] constructs tours by iteratively selecting the closest unvisited city. Zuhanda et al. [13] demonstrated that hybrid methods consistently outperform pure heuristics on TSPLIB benchmarks. Alkafaween et al. [14] proposed an iterative approximate method achieving polynomial-time solutions with competitive quality.

Local search operators refine initial tours through edge exchanges. The k-opt family [15], [16], particularly 2-opt and 3-opt, systematically improves tours by removing and reconnecting edges. The Lin-Kernighan-Helsgaun algorithm [17], [18] represents the state-of-the-art, employing variable k-opt moves with intelligent candidate selection to achieve near-optimal solutions on large instances.

B. Metaheuristic Approaches

Metaheuristic algorithms escape local optima through diversified search mechanisms. Genetic algorithms [19] employ evolutionary principles, with Al-Ibrahim [20] demonstrating near-optimal solutions through specialized crossover operators. Bio-inspired methods have shown promise: Abd Algani [21] applied African Buffalo Optimization to TSP, while Ant Colony Optimization [22] uses pheromone-based search strategies.

Swarm intelligence techniques offer efficient optimization across domains. Kou and Wei [23] combined particle swarm optimization with grey wolf algorithm for improved convergence. Wu et al. [24] developed dynamic multi-population PSO for complex trajectory planning. Recent applications by Zhou [25] and Bai [26] to resource allocation problems demonstrate the versatility of these approaches.

C. Learning-Based Approaches

Deep learning has introduced end-to-end TSP solving. Kafakthong and Sinapiromsaran [27] proposed deep attention mechanisms predicting near-optimal tours directly from distance matrices. Pointer Networks [28] and actor-critic architectures [29] generate tours without supervised training. Graph Neural Networks [30] and Transformers [31] capture complex inter-node relationships. However, these approaches typically require substantial GPU resources and extensive training, limiting their accessibility for resource-constrained environments.

RL addresses sequential decision-making in combinatorial optimization. Wang et al. [32] showed Double Q-Learning outperforms traditional Q-Learning and SARSA for TSP. Pan et al. [33] proposed hierarchical RL scaling to instances with thousands of nodes, though computational requirements remain significant.

D. Hybrid Methods

Hybrid approaches combine classical and learning-based techniques. d'O Costa et al. [34] trained policy models to guide 2-opt operations. Zheng et al. [35] integrated Q-learning into LKH, forming Reinforced LKH with strong TSPLIB performance. NeuroLKH [7] uses neural networks to predict edge candidates without modifying LKH's core structure.

El Jaghaoui et al. [36], [37] explored adaptive exploration-exploitation strategies and combined Q-learning with ACO, demonstrating improved convergence on benchmarks. Ruan et al. [38] merged RL with genetic algorithms for enhanced solution discovery.

E. Positioning of this Work

Existing hybrid methods often rely on complex architectures or multi-dimensional Q-tables that scale poorly. Deep learning approaches typically demand GPU acceleration and extensive computational resources, limiting their accessibility for resource-constrained environments. Our approach addresses these limitations through a minimalist one-dimensional Q-table indexed by cities, drastically reducing memory and computational requirements. The method operates efficiently on standard CPU-only hardware without requiring GPUs or specialized equipment. Unlike methods requiring extensive offline training, our framework learns dynamically during optimization. By combining nearest neighbor initialization with RL-guided 2-opt and 3-opt operators, we demonstrate that effective TSP optimization is achievable on commodity hardware across 55 TSPLIB instances.

III. Methodology

We propose a hybrid framework that combines Q-learning with local search methods for solving the TSP. The key contribution is the use of RL to intelligently guide edge selection in 2-opt and 3-opt operators. This section presents the RL fundamentals (Section III-A), classical local search methods (Section III-B), and our proposed RL-2opt and RL-3opt algorithms (Section III-C).

A. Reinforcement Learning Fundamentals

RL [39] is a machine learning paradigm in which an agent learns to make optimal decisions through interaction with an environment. This process is formalized as a Markov Decision Process (MDP), defined by the tuple $\langle S, A, P, R\rangle$, where S is the state space, A the action space, P the transition probabilities, and R the reward function. The agent seeks to maximize cumulative reward by learning an optimal policy $\pi : S \to A$. Q-learning is a model-free RL algorithm that learns the Q-function, representing the expected return of taking action a in state s. The Q-values are updated using the Bellman equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right], \quad (2)$$

where $\alpha \in (0,1]$ is the learning rate, $\gamma \in [0,1]$ the discount factor, r the immediate reward, and s' the resulting state.

B. Local Search Methods

Local search methods iteratively improve solutions by exploring neighborhoods through local modifications. For the TSP, *k*-opt operators [16] represent the most prominent local search techniques, where *k* denotes the number of edges removed and reconnected.

The 2-opt operator removes two non-adjacent edges and reconnects the path segments in an alternative configuration. With $O(n^2)$ complexity per iteration, 2-opt quickly improves solutions but often converges to mediocre local optima [15].

The 3-opt operator extends this by removing three edges and evaluating multiple reconnection patterns [16]. While producing superior solutions, its $O(n^3)$ complexity becomes prohibitive for large instances. Section III-C presents our RL-guided approach that addresses this computational bottleneck.

C. RL-Guided Local Search Framework

Given an initial TSP tour, we apply a Q-learning-guided local search procedure to iteratively refine the solution. This framework integrates RL with classical k-opt operators to intelligently guide edge selection. The following sections present two algorithms: RL-2opt (Section III-C1), which combines Q-learning with 2-opt, and RL-3opt (Section III-C2), which extends the approach to 3-opt operators.

1) RL-2opt algorithm: We propose an RL-guided variant of the 2-opt algorithm in which a Q-learning agent learns to identify promising city pairs for edge swaps. Unlike classical 2-opt, which exhaustively examines all possible swaps, our approach prioritizes cities that are more likely to yield improvements based on accumulated experience.

a) Q-Learning Components: The RL framework is defined as follows:

- State: The current tour T, represented as an ordered sequence of cities.
- Action: Select two non-adjacent cities (i, j) and perform a 2-opt swap by reversing the tour segment between them.
- Reward: $r = \text{len}_{\text{current}} - \text{len}_{\text{new}}$, where $\text{len}_{\text{current}}$ and len_{new} are the tour lengths before and after the swap. The reward is positive if the swap shortens the tour; otherwise, $r = 0$ and the swap is rejected.
- Q-Table: A one-dimensional array $Q \in \mathbb{R}^n$, where each entry $Q[i]$ estimates the expected benefit when city i participates in a swap. This compact representation avoids the computational overhead of a full state-action Q-table.

b) Action Selection Strategy: City selection follows an ε-greedy policy that balances exploration and exploitation:

- Exploration (probability ε): Select two distinct cities (i, j) uniformly at random.
- Exploitation (probability $1 - \varepsilon$): Choose the city with the highest Q-value, $i = \arg\max_k Q[k]$, then select a second city $j \neq i$ randomly.

The underlying intuition is that well-positioned cities yield minimal improvement when swapped, maintaining low Q-values. Conversely, poorly positioned cities produce significant gains when swapped, accumulating higher Q-values over time. This mechanism naturally directs the search toward unstable tour regions where improvements are most likely.

c) Efficient Tour Update: The 2-opt swap removes edges $(i, \text{succ}(i))$ and $(j, \text{succ}(j))$ and adds edges (i, j) and $(\text{succ}(i), \text{succ}(j))$, where $\text{succ}(i)$ denotes the successor of city i in tour T. The tour length change can be computed locally without recalculating the entire tour:

$$\text{dist}_{\text{removed}} = d(i, \text{succ}(i)) + d(j, \text{succ}(j)), \quad (3)$$

$$\text{dist}_{\text{added}} = d(i, j) + d(\text{succ}(i), \text{succ}(j)), \quad (4)$$

$$\text{len}_{\text{new}} = \text{len}_{\text{current}} - \text{dist}_{\text{removed}} + \text{dist}_{\text{added}}. \quad (5)$$

This localized computation ensures $O(1)$ update complexity per swap, critical for scalability to large instances.

d) Q-Value Update: After each swap attempt, the Q-values of the involved cities are updated using a simplified temporal difference rule:

$$Q[i] \leftarrow Q[i] + \alpha(r - Q[i]), \quad Q[j] \leftarrow Q[j] + \alpha(r - Q[j]), \quad (6)$$

where $\alpha \in (0,1]$ is the learning rate. No discount factor γ is used since each iteration involves a single immediate decision with no delayed consequences, eliminating the need for multi-step credit assignment. The complete procedures are formalized in Algorithms 1 and 2.

Algorithm 1 2-opt Swap Operation

Input: T: current tour, d: distance matrix, $\text{len}_{\text{current}}$: tour length, i, j: cities to swap
Output: T_{new}: updated tour, len_{new}: new tour length

function SWAP-2OPT($T, d, \text{len}_{\text{current}}, i, j$)
  $\text{dist}_{\text{removed}} \leftarrow d(i, \text{succ}(i)) + d(j, \text{succ}(j))$
  $\text{dist}_{\text{added}} \leftarrow d(i, j) + d(\text{succ}(i), \text{succ}(j))$
  if $\text{dist}_{\text{added}} \geq \text{dist}_{\text{removed}}$ **then**
    return $T, \text{len}_{\text{current}}$ ▷ No improvement
  else
    $\text{len}_{\text{new}} \leftarrow \text{len}_{\text{current}} - \text{dist}_{\text{removed}} + \text{dist}_{\text{added}}$
    Reverse tour segment between $\text{succ}(i)$ and j to obtain T_{new}
    return $T_{\text{new}}, \text{len}_{\text{new}}$
  end if
end function

Algorithm 2 RL-2opt Algorithm

Input: n: number of cities, d: distance matrix, T_{init}: initial tour, len_{init}: initial length, α: learning rate, ε: exploration rate, n_{episodes}: number of episodes
Output: T_{best}: best tour found, len_{best}: best length

function RL-2OPT($n, d, T_{\text{init}}, \text{len}_{\text{init}}, \alpha, \varepsilon, n_{\text{episodes}}$)
  $T_{\text{best}} \leftarrow T_{\text{init}}, \text{len}_{\text{best}} \leftarrow \text{len}_{\text{init}}$
  $Q \leftarrow \mathbf{0}_n$ ▷ Initialize Q-table
  for episode $= 1$ to n_{episodes} **do**
    Select cities (i, j) **using** ε**-greedy:**
    if random() $< \varepsilon$ **then** ▷ Exploration
      $i, j \leftarrow$ two random distinct cities
    else ▷ Exploitation
      $i \leftarrow \arg\max_k Q[k]$
      $j \leftarrow$ random city with $j \neq i$
    end if
    $T_{\text{new}}, \text{len}_{\text{new}} \leftarrow \text{Swap-2opt}(T_{\text{best}}, d, \text{len}_{\text{best}}, i, j)$
    $r \leftarrow \text{len}_{\text{best}} - \text{len}_{\text{new}}$
    Update Q-values:
    $Q[i] \leftarrow Q[i] + \alpha(r - Q[i])$
    $Q[j] \leftarrow Q[j] + \alpha(r - Q[j])$
    if $r > 0$ **then** ▷ Accept improvement
      $T_{\text{best}} \leftarrow T_{\text{new}}, \text{len}_{\text{best}} \leftarrow \text{len}_{\text{new}}$
    end if
  end for
  return $T_{\text{best}}, \text{len}_{\text{best}}$
end function

2) RL-3opt algorithm: We extend the RL-guided approach to the more powerful 3-opt operator, which explores a significantly larger neighborhood than 2-opt. While 2-opt removes two edges and performs a single reconnection, 3-opt removes three edges and evaluates multiple reconnection patterns, potentially yielding higher-quality solutions at the cost of increased computational complexity.

a) Q-Learning Components: The RL framework for 3-opt follows the same structure as RL-2opt, with the key difference being the action space:

- State: The current tour T.
- Action: Select three cities $i < j < k$ in tour T, remove the edges $(i, \text{succ}(i))$, $(j, \text{succ}(j))$, and $(k, \text{succ}(k))$, and evaluate all valid reconnection patterns to find the best configuration.
- Reward: $r = \text{dist}_{\text{removed}} - \text{dist}_{\text{added}}(m^*)$, where m^* is the best reconnection pattern among the seven non-trivial configurations.
- Q-Table: The same one-dimensional array $Q \in \mathbb{R}^n$ as in RL-2opt, where $Q[i]$ estimates the expected benefit when city i is involved in a 3-opt move.

b) 3-opt Reconnection Patterns: Removing three edges $(i, \text{succ}(i))$, $(j, \text{succ}(j))$, and $(k, \text{succ}(k))$ partitions the tour into three segments:

- S_1: from $\text{succ}(i)$ to j,
- S_2: from $\text{succ}(j)$ to k,
- S_3: from $\text{succ}(k)$ to i.

Each segment can be traversed forward or reversed, yielding $2^3 = 8$ total combinations. The first configuration (all segments in original orientation) is trivial and yields no change. The remaining seven configurations are evaluated. Table I lists all eight cases and their corresponding edge insertions.

TABLE I. THE 8 RECONNECTION PATTERNS IN 3-OPT AND THEIR INSERTED EDGES. CONFIGURATION 1 IS THE IDENTITY AND IS IGNORED. NOTATION: $^\circlearrowleft$ INDICATES SEGMENT REVERSAL

Case	Segment orientation	Inserted edges
1	$S_1 \to S_2 \to S_3$	Identity (ignored)
2	$S_1^\circlearrowleft \to S_2 \to S_3$	$(i, j), (\text{succ}(i), \text{succ}(j)), (k, \text{succ}(k))$
3	$S_1 \to S_2^\circlearrowleft \to S_3$	$(i, \text{succ}(i)), (j, k), (\text{succ}(j), \text{succ}(k))$
4	$S_1 \to S_2 \to S_3^\circlearrowleft$	$(i, \text{succ}(i)), (j, \text{succ}(j)), (k, i)$
5	$S_1^\circlearrowleft \to S_2^\circlearrowleft \to S_3$	$(i, j), (\text{succ}(i), k), (\text{succ}(j), \text{succ}(k))$
6	$S_1^\circlearrowleft \to S_2 \to S_3^\circlearrowleft$	$(i, k), (\text{succ}(i), \text{succ}(j)), (j, \text{succ}(k))$
7	$S_1 \to S_2^\circlearrowleft \to S_3^\circlearrowleft$	$(i, \text{succ}(i)), (j, k), (\text{succ}(j), i)$
8	$S_1^\circlearrowleft \to S_2^\circlearrowleft \to S_3^\circlearrowleft$	$(i, k), (\text{succ}(i), j), (\text{succ}(j), \text{succ}(k))$

c) Efficient Configuration Evaluation: To avoid recomputing the full tour length for each configuration, we use local distance calculations:

$$\text{dist}_{\text{removed}} = d(i, \text{succ}(i)) + d(j, \text{succ}(j)) + d(k, \text{succ}(k)), \quad (7)$$

$$\text{dist}_{\text{added}}(m) = d(y_1) + d(y_2) + d(y_3), \quad (8)$$

where (y_1, y_2, y_3) are the three edges inserted in configuration $m \in \{2, \ldots, 8\}$. The best configuration m^* is:

$$m^* = \arg \min_{m \in \{2,\ldots,8\}} \text{dist}_{\text{added}}(m). \quad (9)$$

If $\text{dist}_{\text{added}}(m^*) < \text{dist}_{\text{removed}}$, the move improves the tour and is accepted:

$$\text{len}_{\text{new}} = \text{len}_{\text{current}} - \text{dist}_{\text{removed}} + \text{dist}_{\text{added}}(m^*), \quad (10)$$

$$r = \text{dist}_{\text{removed}} - \text{dist}_{\text{added}}(m^*). \quad (11)$$

Otherwise, the tour remains unchanged and $r = 0$.

d) Q-Value Update: After each move attempt, the Q-values of all three involved cities are updated:

$$Q[c] \leftarrow Q[c] + \alpha(r - Q[c]), \quad \forall c \in \{i, j, k\}. \quad (12)$$

e) Action Selection Strategy: City selection follows the same ε-greedy policy as RL-2opt: with probability ε (exploration), we select three distinct cities i, j, k uniformly at random; with probability $1 - \varepsilon$ (exploitation), we choose $i = \arg\max_c Q[c]$, then randomly select two additional distinct cities $j \neq i$ and $k \neq i, j$. The complete procedures are formalized in Algorithms 3 and 4.

Algorithm 3 3-opt Swap Operation

Input: T: current tour, d: distance matrix, $\text{len}_{\text{current}}$: tour length, i, j, k: cities with $i < j < k$
Output: T_{new}: updated tour, len_{new}: new tour length

function SWAP-3OPT($T, d, \text{len}_{\text{current}}, i, j, k$)
  $\text{dist}_{\text{removed}} \leftarrow d(i, \text{succ}(i)) + d(j, \text{succ}(j)) + d(k, \text{succ}(k))$
  $r_{\text{best}} \leftarrow 0$
  $T_{\text{new}} \leftarrow T, \text{len}_{\text{new}} \leftarrow \text{len}_{\text{current}}$
  for each configuration $m \in \{2, \ldots, 8\}$ **do**
    $(y_1, y_2, y_3) \leftarrow$ edges inserted in configuration m (from Table I)
    $\text{dist}_{\text{added}} \leftarrow d(y_1) + d(y_2) + d(y_3)$
    $r \leftarrow \text{dist}_{\text{removed}} - \text{dist}_{\text{added}}$
    if $r > r_{\text{best}}$ **then**
      $r_{\text{best}} \leftarrow r$
      $\text{len}_{\text{new}} \leftarrow \text{len}_{\text{current}} - \text{dist}_{\text{removed}} + \text{dist}_{\text{added}}$
      $T_{\text{new}} \leftarrow$ tour reassembled according to configuration m
    end if
  end for
  return $T_{\text{new}}, \text{len}_{\text{new}}$
end function

Having presented the detailed mechanisms of both RL-2opt and RL-3opt algorithms, Fig. 1 provides a unified visual representation of the complete framework.

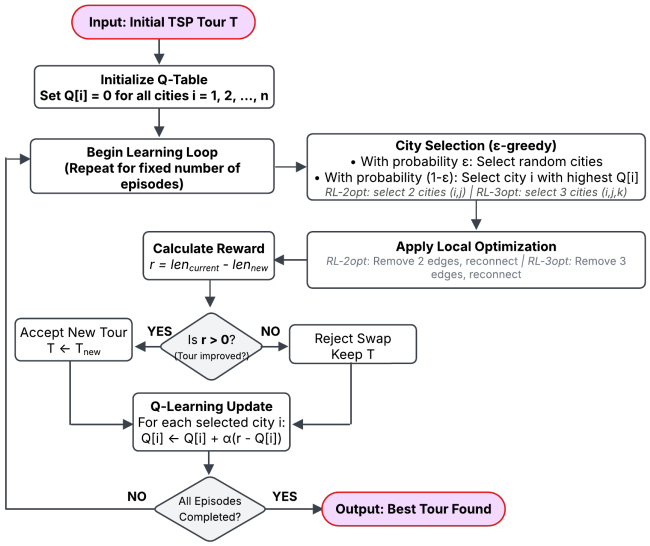

Fig. 1. Flowchart of the Q-learning guided local search framework.

Algorithm 4 RL-3opt Algorithm

Input: n: number of cities, d: distance matrix, T_{init}: initial tour, len_{init}: initial length, α: learning rate, ε: exploration rate, n_{episodes}: number of episodes
Output: T_{best}: best tour found, len_{best}: best length

function RL-3OPT($n, d, T_{\text{init}}, \text{len}_{\text{init}}, \alpha, \varepsilon, n_{\text{episodes}}$)
  $T_{\text{best}} \leftarrow T_{\text{init}}, \text{len}_{\text{best}} \leftarrow \text{len}_{\text{init}}$
  $Q \leftarrow \mathbf{0}_n$ ▷ Initialize Q-table
  for episode $= 1$ to n_{episodes} **do**
    Select cities (i, j, k) **using** ε**-greedy:**
    if random() $< \varepsilon$ **then** ▷ Exploration
      $i, j, k \leftarrow$ three random distinct cities
    else ▷ Exploitation
      $i \leftarrow \arg\max_c Q[c]$
      $j, k \leftarrow$ two random distinct cities with $j \neq i, k \neq i, j \neq k$
    end if
    $T_{\text{new}}, \text{len}_{\text{new}} \leftarrow$ Swap-3opt($T_{\text{best}}, d, \text{len}_{\text{best}}, i, j, k$)
    $r \leftarrow \text{len}_{\text{best}} - \text{len}_{\text{new}}$
    Update Q-values:
    for $c \in \{i, j, k\}$ **do**
      $Q[c] \leftarrow Q[c] + \alpha(r - Q[c])$
    end for
    if $r > 0$ **then** ▷ Accept improvement
      $T_{\text{best}} \leftarrow T_{\text{new}}, \text{len}_{\text{best}} \leftarrow \text{len}_{\text{new}}$
    end if
  end for
  return $T_{\text{best}}, \text{len}_{\text{best}}$
end function

IV. EXPERIMENTAL EVALUATION

This section presents the experimental protocol, parameter configuration, and comprehensive performance analysis of the proposed RL-guided local search framework.

A. Experimental Setup

a) Implementation and Hardware: All algorithms were implemented in C and executed on an HP laptop equipped with an Intel® Core™ i5-8265U CPU (1.6 GHz base frequency) and 16 GB of RAM. No parallelization or GPU acceleration was used, ensuring a fair comparison based on single-threaded performance.

b) Benchmark Instances: We evaluate our approach on 55 symmetric instances from the TSPLIB repository [11], ranging from 51 to 18,512 cities. All instances use the Euclidean distance metric (`EUC_2D`), ensuring consistency in distance calculations across experiments. Table II lists all tested instances with their optimal solutions.

c) Experimental Protocol: For each instance, the following three-stage workflow is applied:

1) Initialization: Generate an initial tour using the Nearest Neighbor (NN) heuristic starting from a randomly selected city.
2) Refinement: Apply RL-2opt or RL-3opt independently to improve the initial tour. Each RL algorithm is executed for 10,000 episodes.

3) Baseline Comparison: Evaluate against classical 2-opt and 3-opt local search using the same initial tour to ensure fair comparison.

To ensure statistical reliability, each RL algorithm is executed 20 independent runs per instance with different random seeds. We report the best solution across these runs, along with a variability metric (GAP%) defined as:

$$\text{GAP}\% = 100 \times \frac{\sqrt{\frac{1}{20}\sum_{i=1}^{20}(\text{len}_i - \overline{\text{len}})^2}}{\text{len}_{\min}}, \tag{13}$$

where len_i is the tour length in run i, $\overline{\text{len}}$ is the average length, and $\text{len}_{\min}$ is the best length observed. Lower GAP% values indicate more consistent performance across runs.

d) Hyperparameters: All RL experiments use fixed hyperparameters to ensure reproducibility: number of episodes $n_{\text{episodes}} = 10{,}000$, learning rate $\alpha = 0.1$, discount factor $\gamma = 0$ (immediate rewards only), and exploration rate $\varepsilon = 0.1$. These values remain constant across all instances.

e) Performance Metrics: We evaluate algorithms using three metrics:

- Solution Quality: Gap percentage from known optimal or best-known solutions:

$$\text{Gap}(\%) = \frac{\text{Solution} - \text{Optimal}}{\text{Optimal}} \times 100. \tag{14}$$

- Computational Time: Wall-clock execution time in seconds, excluding I/O operations.
- Stability: Variability across runs measured by GAP% as shown in Equation (13).

B. Results and Analysis

Table II presents comprehensive results for all 55 instances, including optimal values, tour lengths, computation times, and variability metrics for each method. Classical 3-opt becomes computationally impractical for the largest instances (indicated by —), requiring estimated days of computation time.

Before analyzing these results in detail, we contextualize our approach relative to existing TSP methods. Specialized solvers like LKH [17], [18] represent the state of the art for solution quality. We compared our approach with LKH version 3.0.6 (default parameters, run until convergence), which systematically produces the highest-quality solutions across all instances. However, execution time differences are particularly pronounced on large instances: while LKH systematically achieves the highest quality solutions through extensive computation (hours), our framework provides near-optimal results orders of magnitude faster (seconds). This confirms our positioning: our framework is not designed to replace specialized solvers like LKH, but rather to provide a practical, resource-efficient alternative under constrained computational conditions—suitable for real-time applications, standard hardware deployment, or batch processing of thousands of instances.

Other Q-learning approaches for TSP focus on tour construction by learning city selection policies, while our method addresses a fundamentally different problem: *improving* existing tours rather than *generating* tours from scratch. Classical Q-learning tour construction methods typically achieve gaps of 5–15%, while our improvement-focused approach delivers sub-1% gaps on small instances (≤ 100 cities). Direct experimental comparison is not included because these approaches operate in fundamentally different settings: construction methods learn sequential decisions from an empty tour, while our improvement approach starts from an existing solution and learns neighborhood restriction. Comparing final tour quality would conflate these distinct problem formulations (generation vs. refinement) and initial conditions rather than isolating the contribution of RL guidance itself.

We compare four local search improvement methods across all instances: classical 2-opt, classical 3-opt, RL-guided 2-opt (RL-2opt), and RL-guided 3-opt (RL-3opt). Classical 2-opt achieves an average optimality gap of 8.95% across all problem sizes, while classical 3-opt reduces this to 5.31% on instances where it is computationally feasible. RL-guided methods demonstrate size-dependent performance: For instances up to 1,000 cities, RL-3opt achieves exceptional quality with gaps of 0.9–2.2%, including near-optimal solutions on several instances (`rd100`: 0.00%, `kroA100`: 0.01%, `berlin52`: 0.03%), while RL-2opt maintains competitive gaps of 1.6–2.6%. For instances between 1,000–5,000 cities, performance degrades: RL-3opt achieves 6.3% gaps compared to 5.5% for classical 3-opt, while RL-2opt reaches 9.1% gaps comparable to classical 2-opt's 9.0%. Despite this quality trade-off in the 1,000–5,000 range, RL methods maintain substantial computational advantages with speedups exceeding 100× to 30,000×. For very large instances exceeding 5,000 cities, RL-3opt achieves 9.8% gaps while RL-2opt reaches 12.2%. In this range, RL-3opt matches classical 2-opt quality (both 9.8%) while executing orders of magnitude faster, though RL-2opt shows some quality degradation compared to classical methods.

Table III reveals systematic performance differences across improvement methods.For instances up to 1,000 cities, RL-3opt achieves 2-4× better gaps than classical 3-opt. However, for instances between 1,000-5,000 cities, classical 3-opt achieves slightly better quality (5.5% vs 6.3%) while RL-3opt maintains substantial computational advantages.

Fig. 2 illustrates these quality differences on five representative instances spanning different problem sizes. RL-3opt (dark blue) achieves near-optimal solutions, including exact optimality on `rd100`, while classical 2-opt (dark red) and classical 3-opt (light red) consistently show higher approximation ratios. RL-2opt (light blue) provides intermediate quality between classical and RL-3opt approaches.

Beyond quality, computational efficiency differs dramatically across methods. Classical 3-opt exhibits cubic scaling, becoming impractical beyond 1000 cities (requiring 16 minutes to over six hours for instances with 2,000–6,000 cities) and entirely infeasible for problems exceeding 10,000 cities. Classical 2-opt executes faster but still requires seconds for large instances. In contrast, RL-guided methods maintain near-linear scalability: instances up to 2,400 cities complete in under 0.25 seconds, up to 6,000 cities finish in under 1 second, and 18,512-city problems solve in under 4 seconds—yielding speedups over classical 3-opt ranging from 4,334× to 32,323×.

TABLE II. COMPREHENSIVE COMPARISON ON 55 TSPLIB INSTANCES. FOR EACH METHOD: BEST TOUR LENGTH AND RUNTIME (SECONDS). FOR RL METHODS: GAP% STABILITY METRIC OVER 20 RUNS. CLASSICAL 3-OPT BECOMES IMPRACTICAL (—) FOR THE LARGEST INSTANCES

Problem	Optimal	NN	NN+2opt		NN+3opt		NN+RL2opt			NN+RL3opt		
			Dist	Time (s)	Dist	Time (s)	Best	GAP%	Time (s)	Best	GAP%	Time (s)
kroA100	21282	26856	23228	0.001	21864	0.033	21371	0.50	0.015	21285	0.26	0.021
kroB100	22141	29155	22907	0.002	22886	0.0130	22295	0.45	0.014	22243	0.22	0.022
kroC100	20749	26327	22943	0.001	22176	0.009	20947	0.42	0.016	20852	0.29	0.021
kroD100	21294	26950	23422	0.002	21826	0.015	21845	0.49	0.015	21485	0.33	0.021
kroE100	22068	27587	24475	0.001	23490	0.014	22403	0.40	0.014	22141	0.40	0.023
kroA200	29368	35798	31735	0.002	29823	0.260	29953	0.26	0.022	29963	0.27	0.025
kroB200	29437	36982	32539	0.003	31124	0.232	30205	0.51	0.021	30154	0.38	0.028
rd100	7910	9941	8731	< 0.001	8207	0.025	7975	0.34	0.016	7910	0.12	0.021
rd400	15281	19168	16847	0.004	15870	3.212	15745	0.30	0.037	15676	0.19	0.045
ch130	6110	7575	6848	0.001	6459	0.054	6284	0.31	0.017	6239	0.23	0.023
ch150	6528	8195	6727	0.001	6812	0.060	6597	0.39	0.017	6580	0.11	0.024
berlin52	7542	8981	8287	< 0.001	7801	0.002	7544	0.11	0.012	7544	0.10	0.019
eil51	426	514	449	< 0.001	437	0.002	432	0.09	0.011	430	0.09	0.018
eil76	538	712	572	< 0.001	552	0.010	549	0.11	0.012	545	0.13	0.019
st70	675	806	727	< 0.001	689	0.009	696	0.18	0.012	687	0.07	0.022
eil101	629	825	707	0.001	669	0.025	657	0.19	0.017	656	0.15	0.020
pr76	108159	153462	119489	< 0.001	109170	0.015	109210	0.33	0.014	109120	0.25	0.020
pr107	44303	46678	46194	0.001	44728	0.021	44768	0.41	0.017	44724	0.28	0.020
pr124	59030	69299	64335	0.002	60484	0.032	59685	0.39	0.016	59551	0.31	0.022
pr152	73682	85703	77351	0.002	77536	0.035	74004	0.20	0.018	73877	0.17	0.027
pr226	80369	94685	85113	0.002	81594	0.217	80914	0.27	0.023	80769	0.24	0.029
pr264	49135	58023	55188	0.003	50279	0.643	50803	0.57	0.024	50679	0.43	0.033
pr299	48191	59899	51414	0.003	49467	0.74	49955	0.60	0.029	49653	0.52	0.036
pr439	107217	131282	117331	0.003	113722	2.409	110231	0.87	0.039	109928	0.61	0.043
pr1002	259045	315597	284317	0.028	272929	36.104	275363	0.64	0.081	272280	0.34	0.090
pr2392	378032	461207	417036	0.456	401880	957.91	417473	0.37	0.206	406317	0.42	0.221
rat99	1211	1565	1331	< 0.001	1306	0.024	1272	0.17	0.015	1260	0.10	0.025
rat195	2323	2762	2511	0.001	2419	0.296	2387	0.28	0.020	2386	0.18	0.027
rat575	6773	8449	7342	0.011	7231	11.596	7085	0.57	0.045	7042	0.47	0.052
rat783	8806	11255	9816	0.014	9308	16.505	9296	0.27	0.067	9219	0.31	0.087
u159	42080	54669	47578	0.001	44916	0.076	43583	0.41	0.018	42900	0.39	0.024
u1060	224094	281636	247562	0.062	237724	63.38	241904	0.89	0.085	233538	0.80	0.111
u1432	152970	188815	168415	0.099	163043	120.598	167796	0.52	0.110	164102	0.36	0.132
u1817	57201	71103	62867	0.147	61265	437.57	63969	0.44	0.145	61995	0.34	0.178
u2152	64253	80180	70741	0.287	68741	681.072	71870	0.33	0.168	70187	0.27	0.191
u2319	234256	278783	246510	0.257	243418	702.340	254347	0.34	0.178	243379	0.24	0.203
lin105	14379	20363	16400	0.001	15394	0.040	14630	0.47	0.016	14529	0.20	0.019
rl1304	252948	339797	284062	0.140	273043	179.664	274694	0.85	0.136	269556	0.89	0.138
rl1323	270199	332095	298286	0.075	286679	182.035	284152	0.54	0.101	280689	0.48	0.122
rl1889	316536	400685	343363	0.144	332865	507.93	344585	0.42	0.151	334100	0.50	0.175
rl5915	565530	707499	622550	2.501	588813	22625.876	619163	0.25	0.669	606306	0.23	0.700
rl5934	556045	683806	614093	3.122	597867	18309.009	623758	0.22	0.667	606867	0.35	0.705
rl11849	923288	1139496	1014854	21.466	—	—	1025936	0.30	1.888	1006586	0.17	2.00
bier127	118282	135752	123768	0.001	120764	0.099	120462	0.56	0.016	119852	0.91	0.023
vm1084	239297	301469	258092	0.074	250083	129.744	258659	0.79	0.085	254407	0.55	0.109
vm1748	336556	408089	365952	0.273	353458	473.991	363160	0.86	0.133	355258	0.46	0.163
pcb3038	137694	175573	152112	0.544	147145	2107.079	151471	0.27	0.263	147235	0.26	0.280
fl3795	28772	34226	30401	1.399	29713	3784.776	32669	0.46	0.371	31037	0.92	0.397
fnl4461	182566	227157	200121	1.522	192642	8661.879	201355	0.20	0.451	195775	0.22	0.475
usa13509	19982859	25047673	21984724	28.548	—	—	22777155	0.15	2.31	22181451	0.14	2.73
brd14051	469385	577037	514523	22.656	—	—	535127	0.18	2.50	522622	0.11	2.90
d493	35002	43646	37848	0.0160	36373	6.386	36337	0.43	0.044	36112	0.28	0.052
d2103	80450	87469	83437	0.208	81771	371.815	85468	1.01	0.187	84925	0.61	0.207
d15112	1573084	1948433	1719624	26.630	—	—	1760464	1.39	3.66	1731045	0.65	3.84
d18512	645238	797562	703839	45.482	—	—	730690	0.24	3.82	716002	0.13	3.94

TABLE III. AVERAGE OPTIMALITY GAPS (%) BY INSTANCE SIZE, COMPUTED AS GAP(%) = (SOLUTION - OPTIMAL) / OPTIMAL × 100

Size (cities)	N	2-opt	3-opt	RL-2opt	RL-3opt
≤ 100	(12)	8.7	3.8	1.6	**0.9**
100–1,000	(20)	9.0	4.3	2.6	**2.2**
1,000–5,000	(16)	9.0	5.5	9.1	**6.3**
$> 5{,}000$	(7)	9.8	—	12.2	**9.8**

The combined quality-time advantage varies by problem size: small instances ($\leq$100 cities) see RL-3opt achieving 4× better quality than 3-opt with comparable execution times, medium instances (100–1,000 cities) combine millisecond execution with 3× better quality than 3-opt, and large instances (1,000–5,000 cities) demonstrate RL-3opt transforming hours-long searches into sub-second optimizations with 2× quality improvement over 3-opt and 4× over 2-opt. Very large instances (>5,000 cities) remain infeasible for classical 3-opt while RL methods scale gracefully: RL-3opt matches classical 2-opt quality but executes orders of magnitude faster.

Fig. 3 visualizes these advantages. Panel (a) demonstrates RL methods scaling to 18,000+ cities in under 4 seconds, while classical 3-opt exhibits cubic growth. Panel (b) confirms Pareto dominance: classical 2-opt offers speed without quality (ratios 1.08–1.11), classical 3-opt offers better quality at prohibitive cost, while RL methods provide the fastest solution for any

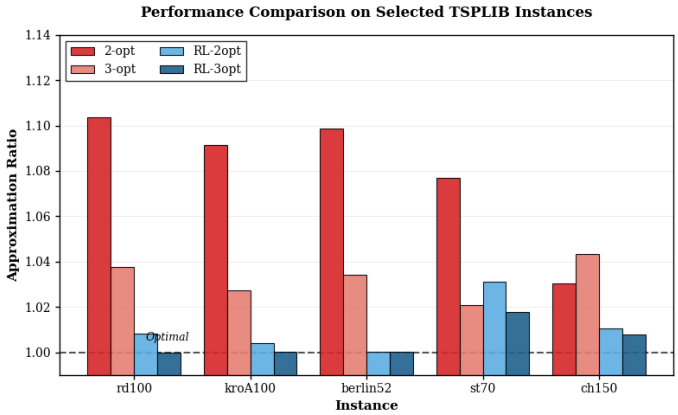

Fig. 2. Performance comparison on five representative instances. RL-3opt (dark blue) achieves near-optimal solutions including exact optimality on `rd100`. RL-2opt (light blue) provides intermediate quality. Classical methods (red) show significantly higher approximation ratios.

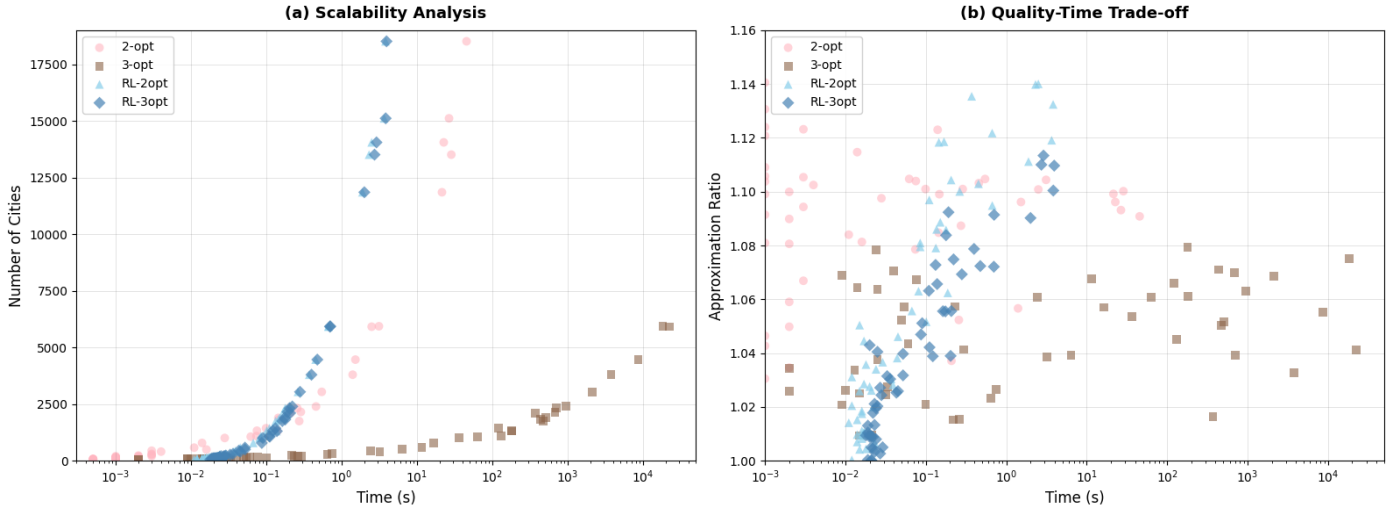

Fig. 3. Computational performance on 55 TSPLIB instances. (a) RL methods maintain near-linear scaling while classical 3-opt exhibits cubic explosion. (b) RL methods dominate the Pareto frontier with approximation ratios of 1.01–1.12 and execution times under 5 seconds.

desired quality level.

To assess robustness, we measured variability across 20 independent runs using the GAP% stability metric (Equation 13). Both RL methods demonstrate excellent stability with GAP% values of 0.1–1.0%. Best cases include `eil51` (0.09%), `usa13509` (0.14–0.15%), and `brd14051` (0.11–0.18%), confirming reliable Q-learning convergence suitable for production deployment.

C. Discussion

Our results demonstrate that a simple one-dimensional Q-table can effectively guide local search for TSP, but with clear limitations tied to problem size. The approach works well because it learns which cities are poorly positioned during optimization and focuses computational effort there. On instances up to 1,000 cities, this adaptive strategy produces solutions 2–4× better than classical 3-opt while running much faster. The Q-values act as a priority signal, directing the algorithm toward tour segments most likely to benefit from modification.

Performance declines on larger instances because the learning signal becomes weaker. With 10,000 fixed episodes, each city receives less attention as n grows, making it harder to identify and fix problematic regions. This reveals the core trade-off of our design: we sacrificed representational power (one-dimensional Q-table instead of multi-dimensional) to gain computational efficiency and simplicity. Methods like Reinforced LKH [35] maintain quality on large problems through richer state representations, while neural approaches [31], [10] learn powerful policies but require extensive training on GPUs. Our framework sits between these extremes—it learns during optimization rather than through offline training, making it accessible but limiting its ability to handle very large instances.

The dramatic speedups (401–32,323×) come from a fundamental algorithmic change: sampling a constant number of moves instead of exhaustively examining all $O(n^3)$ possibilities. This matters practically. A logistics company optimizing 500 delivery routes can complete the task in seconds rather than minutes, enabling real-time adjustments as conditions change. For applications where speed and accessibility matter

more than guaranteed optimality, this framework offers a compelling option. However, the 6.3% gap on medium instances (1,000–5,000 cities) versus 5.5% for classical 3-opt shows there are situations where the extra computation time is worth it.

The broader contribution is accessibility. Running on standard CPUs without hyperparameter tuning makes high-quality TSP optimization available to organizations without specialized computing infrastructure. This democratization matters for small businesses, educational institutions, and resource-constrained environments.

V. Conclusion

This paper introduced a Q-learning-guided local search framework that addresses neighborhood explosion in classical TSP methods by learning city-specific Q-values to guide exploration toward promising moves. The approach transforms $O(n^3)$ 3-opt complexity into practical sub-second optimization.

Experimental evaluation on 55 TSPLIB instances demonstrates size-dependent performance characteristics. For instances up to 1,000 cities, RL-3opt achieves 0.9–2.2% optimality gaps compared to 3.8–4.3% for classical 3-opt, with speedups exceeding 1,000×. For instances between 1,000–5,000 cities, classical 3-opt achieves better quality (5.5% vs 6.3%), but RL-3opt maintains practical efficiency with speedups reaching 32,323× and sub-second execution. For instances exceeding 5,000 cities, RL-3opt achieves 9.8% gaps matching classical 2-opt quality while executing orders of magnitude faster. Classical 3-opt becomes computationally infeasible beyond 1,000 cities.

By delivering near-optimal solutions on small instances and competitive solutions on large instances using standard CPU hardware without GPU acceleration, this framework democratizes access to effective TSP optimization. The combination of sub-second to few-second execution across all problem sizes, minimal configuration requirements, and favorable quality-time trade-offs makes RL-guided local search a practical alternative for real-world logistics, manufacturing, and routing applications where both solution quality and computational efficiency matter.

Future work will focus on implementing dynamic episode scaling and adaptive learning rates to improve performance on large instances, and extending the framework to related combinatorial problems including Vehicle Routing and Job Shop Scheduling.

References

[1] G. Laporte, "The traveling salesman problem: An overview of exact and approximate algorithms," *European Journal of Operational Research*, vol. 59, no. 2, pp. 231–247, 1992.

[2] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, New York, 1979.

[3] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook, *The Traveling Salesman Problem: A Computational Study*, Princeton, NJ: Princeton University Press, 2011.

[4] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: A methodological tour d'horizon," *European Journal of Operational Research*, vol. 290, no. 2, pp. 405–421, 2021.

[5] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev, "Reinforcement learning for combinatorial optimization: A survey," *Computers & Operations Research*, vol. 134, p. 105400, 2021.

[6] B. Hudson, Q. Li, M. Malencia, and A. Prorok, "Graph neural network guided local search for the traveling salesperson problem," *arXiv preprint arXiv:2110.05291*, 2021.

[7] L. Xin, W. Song, Z. Cao, and J. Zhang, "NeuroLKH: Combining deep learning model with Lin–Kernighan–Helsgaun heuristic for solving the traveling salesman problem," *Advances in Neural Information Processing Systems*, vol. 34, pp. 7472–7483, 2021.

[8] Y.-D. Kwon, J. Choo, B. Kim, I. Yoon, Y. Gwon, and S. Min, "POMO: Policy optimization with multiple optima for reinforcement learning," in *Advances in Neural Information Processing Systems*, vol. 33, pp. 21188–21198, 2020.

[9] M. Kim, J. Park, and J. Park, "Sym-NCO: Leveraging symmetricity for neural combinatorial optimization," *Advances in Neural Information Processing Systems*, vol. 35, pp. 1936–1949, 2022.

[10] C. K. Joshi, T. Laurent, and X. Bresson, "An efficient graph convolutional network technique for the travelling salesman problem," *arXiv preprint arXiv:1906.01227*, 2019.

[11] G. Reinelt, "TSPLIB—A traveling salesman problem library," *ORSA Journal on Computing*, vol. 3, no. 4, pp. 376–384, 1991.

[12] G. Kizilateş and F. Nuriyeva, "On the nearest neighbor algorithms for the traveling salesman problem," in *Advances in Computational Science, Engineering and Information Technology*, Proc. 3rd Int. Conf. on Computational Science, Engineering and Information Technology (CCSEIT-2013), Konya, Turkey, vol. 1, pp. 111–118, Springer, 2013.

[13] M. Zuhanda, H. Mawengkang, S. Efendi, and M. Zarlis, "Hybrid local search algorithm for optimization route of travelling salesman problem," *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 14, no. 9, 2023.

[14] E. Alkafaween, S. Elmougy, E. Essa, S. Mnasri, A. S. Tarawneh, and A. Hassanat, "IAM-TSP: Iterative approximate methods for solving the travelling salesman problem," *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 14, no. 11, 2023.

[15] B. Chandra, H. Karloff, and C. Tovey, "New results on the old k-opt algorithm for the traveling salesman problem," *SIAM Journal on Computing*, vol. 28, no. 6, pp. 1998–2029, 1999.

[16] A. Blazinskas and A. Misevicius, "Combining 2-opt, 3-opt and 4-opt with k-swap-kick perturbations for the traveling salesman problem," Technical Report, Department of Multimedia Engineering, Kaunas University of Technology, 2011.

[17] K. Helsgaun, "An effective implementation of the Lin–Kernighan traveling salesman heuristic," *European Journal of Operational Research*, vol. 126, no. 1, pp. 106–130, 2000.

[18] K. Helsgaun, "An extension of the Lin–Kernighan–Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems," *Roskilde University*, vol. 12, pp. 966–980, 2017.

[19] N. M. Razali and J. Geraghty, "Genetic algorithm performance with different selection strategies in solving TSP," in *Proceedings of the World Congress on Engineering*, vol. 2, no. 1, pp. 1–6, International Association of Engineers, Hong Kong, China, July 2011.

[20] Z. H. Ahmed, "Optimization of the travelling salesman problem using a new hybrid genetic algorithm," *International Journal of Computer Science and Network Security*, vol. 23, no. 10, pp. 189–198, 2020.

[21] Y. M. Abd Algani, "Optimization solutions for solving travelling salesman problem in graph theory using African buffalo mechanism," *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 14, no. 7, pp. 273–282, 2023.

[22] J. Yang, X. Shi, M. Marchese, and Y. Liang, "An ant colony optimization method for generalized TSP problem," *Progress in Natural Science*, vol. 18, no. 11, pp. 1417–1422, 2008.

[23] G. Kou and G. Wei, "Hybrid particle swarm optimization-based modeling of wireless sensor network coverage optimization," *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 14, no. 5, 2023.

[24] R. Wu, Y. Yang, X. Yao, and N. Lu, "Optimal trajectory planning for robotic arm based on improved dynamic multi-population particle swarm optimization algorithm," *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 15, no. 5, 2024.

[25] C. Zhou, "Black widow optimization algorithm for virtual machines migration in the cloud environments," *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 14, no. 8, 2023.

[26] G. Bai, "Enhancing Harris hawks optimization algorithm for resource allocation in cloud computing environments," *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 15, no. 3, 2024.

[27] N. Kafakthong and K. Sinapiromsaran, "Near-optimal traveling salesman solution with deep attention," *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 15, no. 12, pp. 918–928, 2024.

[28] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[29] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," *arXiv preprint arXiv:1611.09940*, 2016.

[30] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[31] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!," *arXiv preprint arXiv:1803.08475*, 2018.

[32] Y. Wang, Z. Zhang, and L. Chen, "Performance comparison of Q-learning, SARSA, and double Q-learning for the traveling salesman problem," *Expert Systems with Applications*, vol. 213, p. 118865, 2023.

[33] X. Pan, Y. Jin, Y. Ding, M. Feng, L. Zhao, L. Song, and J. Bian, "H-TSP: Hierarchically solving the large-scale traveling salesman problem," in *Proc. 37th AAAI Conf. on Artificial Intelligence*, vol. 37, no. 8, pp. 9345–9353, June 2023.

[34] P. R. d'O Costa, J. Rhuggenaath, Y. Zhang, and A. Akcay, "Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning," in *Proceedings of the Asian Conference on Machine Learning*, PMLR, pp. 465–480, Sept. 2020.

[35] J. Zheng, K. He, J. Zhou, Y. Jin, and C. M. Li, "Reinforced Lin–Kernighan–Helsgaun algorithms for the traveling salesman problems," *Knowledge-Based Systems*, vol. 260, p. 110144, 2023.

[36] S. El Jaghaoui, A. K. Elmiad, and A. B. Lmah, "Enhancing the traveling salesman problem solutions with reinforcement learning: A variant exploration-exploitation approach beyond ε-greedy," in *Proc. 14th Int. Conf. on Intelligent Systems: Theories and Applications (SITA)*, IEEE, pp. 1–6, Nov. 2023.

[37] S. El Jaghaoui, A. Benaini, and A. K. Elmiad, "An innovative Q-learning and ACO approaches for the traveling salesman problem," in *International Conference on Logistics Operations Management*, Cham: Springer Nature Switzerland, pp. 166–176, May 2024.

[38] Y. Ruan, Z. Li, and X. Wang, "Combining reinforcement learning and genetic algorithms for the traveling salesman problem," *Applied Intelligence*, vol. 54, pp. 3421–3436, 2024.

[39] P. R. Montague, "Reinforcement learning: an introduction, by Sutton, R. S., & Barto, A. G.," *Trends in Cognitive Sciences*, vol. 3, no. 9, p. 360, 1999.