STATISTICS, OPTIMIZATION AND INFORMATION COMPUTING
Stat., Optim. Inf. Comput., Vol. **00**, March 0000, pp 1–25.
Published online in International Academic Press (www.IAPress.org)

Q-Learning-Assisted Simulated Annealing for Traveling Salesman Problem Optimization

Nouhaila ADIL [1,*], Fakhita EDDAOUDI [1], Halima LAKHBAB [1], Mohamed NAIMI [2]

[1] *Hassan II University of Casablanca, Mathematical Analysis, Algebra and Applications Laboratory, Faculty of Sciences Aïn Chock, Morocco*
[2] *Hassan First University, LAMSAD Laboratory, National School of Applied Sciences, Berrechid, Morocco*

Abstract Simulated Annealing (SA) is a well-established metaheuristic for tackling combinatorial optimization problems. It draws inspiration from the physical process of annealing in metallurgy. In the optimization context, SA iteratively explores the solution space by accepting not only improving solutions but also, with a temperature-dependent probability, non-improving ones. This mechanism enables the algorithm to escape local optima, thereby enhancing its ability to approach the global minimum of an objective function. Nevertheless, its overall performance is susceptible to the choice of the cooling schedule and the use of fixed neighborhood structures. In this work, we include Q-learning into the SA framework to improve its flexibility. Q-learning is a model-free, value-based method that enables an agent to learn optimal action-selection policies by iteratively updating Q-values using rewards obtained through exploration of the environment. The suggested approach directs the search toward more promising areas by dynamically choosing a leader solution from a predefined set of potential solutions that are updated during iterations, using a learned Q-policy. The Q-values are updated according to the relative improvement each leader provides over time, allowing adaptive exploitation of successful guides. Experimental results on popular benchmark instances of the Travelling Salesman Problem (TSP) from TSPLIB95 demonstrate that the Q-learning-guided SA achieves better solution quality compared to classical SA in most of the tested instances. These results demonstrate how experience-driven decision-making in reinforcement learning can enhance metaheuristic performance.

Keywords Combinatorial Optimization, Simulated Annealing, Traveling Salesman Problem, Q-Learning

AMS 2010 subject classifications 90C27, 90B06, 90C59, 90-08, 90C90, 68T05, 68T20

DOI: 10.19139/soic-2310-5070-3028

1. Introduction

Combinatorial optimization problems, such as the Traveling Salesman Problem (TSP), vehicle routing, and scheduling, are notoriously challenging due to their vast solution spaces and the frequent presence of numerous local optima. To address these issues, metaheuristic algorithms have gained widespread use. These methods provide reliable, near-optimal solutions without requiring strong problem-specific assumptions [3, 4]. Among them, SA has retained enduring popularity thanks to its conceptual simplicity, ability to probabilistically escape local minima, and solid theoretical grounding under suitable cooling schedules [5, 7].

Despite these advantages, classical SA has well-known limitations. In complex or high-dimensional search spaces, its reliance on exploring the neighborhood of a single solution at a time can result in insufficient exploration and slow convergence. To overcome these limitations, researchers have increasingly explored hybridization over the past decade to improve SA's performance with adaptive mechanisms that more effectively balance exploration and exploitation, especially in routing contexts [9]. One particularly promising direction is the integration of

*Correspondence to: Nouhaila Adil (Email: adil.nouhaila@gmail.com). Department of Mathematics and Computer Sciences, Hassan II University.

ISSN 2310-5070 (online) ISSN 2311-004X (print)

Reinforcement Learning (RL) techniques, which allow algorithms to dynamically adjust their behavior based on accumulated search experience [23, 6, 10]. By leveraging feedback from past decisions, RL-enhanced metaheuristics can guide the search toward more promising regions of the solution space and have demonstrated competitive performance in routing and scheduling problems.

In this work, we propose *Q-learning-Assisted Simulated Annealing* (QLSA), a hybrid algorithm that integrates a stateless Q-learning mechanism into the SA framework. Unlike conventional SA, which always applies the neighborhood operator to the current solution, QLSA maintains a set of candidate solutions and employs Q-learning to adaptively select which candidate should lead the search. This design allows the algorithm to incorporate historical feedback into the exploration process, thereby improving its ability to escape poor-quality regions and discover better solutions.

Our choice of Q-learning as the reinforcement learning mechanism is motivated by its simplicity, model-free nature, and proven effectiveness in metaheuristic contexts. Q-learning is well-suited for black-box optimization problems where state transitions are not explicitly defined, as it does not require environment modeling or gradient computation, unlike policy-gradient or actor-critic approaches. Furthermore, Q-learning's off-policy nature allows it to learn an action-value function independently of the exploration policy, providing faster convergence and greater stability in static optimization landscapes [8], in contrast to on-policy methods such as SARSA. These properties make Q-learning particularly attractive for integration with SA, where decisions rely on accumulated experience across diverse candidate solutions rather than explicit state modeling.

The contributions of this paper are twofold:

- We propose QLSA, a novel hybrid algorithm that combines Q-learning with SA to enable adaptive candidate selection during the search.
- We evaluate two classical exploration policies; ε-Greedy and Softmax; within QLSA, providing insights into their impact on search performance.

The remainder of this paper is organized as follows. Section 2 reviews related work on hybrid metaheuristics and reinforcement learning-assisted optimization. Sections 3 and 4 define the TSP and detail the proposed QLSA algorithm, respectively. Section 5 presents the experimental setup and results. Finally, Section 6 summarizes our contributions and discusses future research directions.

2. Related work

The Traveling Salesman Problem (TSP) has long been regarded as a cornerstone of operations research and combinatorial optimization. First introduced by Dantzig in 1959 [25], it has continued to draw great interest thanks to its challenging nature and real-world relevance. Renowned for its computational complexity and practical relevance, the TSP is extensively used as a benchmark for evaluating the performance of discrete optimization methods and advancing the limits of existing approaches.

Methods for solving the TSP generally fall into two categories: exact and approximate. Exact approaches guarantee the optimal solution but typically require considerable computational resources and time, making them impractical for large instances. In contrast, approximate methods aim for near-optimal solutions within a reasonable computational effort, often in polynomial time.

Among these approximate approaches, metaheuristics play a particularly prominent role. These high-level strategies are designed to guide the search process through complex solution spaces, often inspired by natural processes or adaptive behaviors. One of their key advantages is their problem-independent nature, allowing them to be applied across a wide range of optimization contexts.

Metaheuristics can be categorized into single-solution and population-based methods. Single solution approaches iteratively refine a single candidate solution, as seen in algorithms like Tabu Search (TS) [26] and Simulated Annealing (SA) [2]. Population-based approaches, on the other hand, explore multiple candidate solutions simultaneously, leveraging collective behaviors for improved search performance. Particle Swarm Optimization (PSO) [27] is a well-known example of this category.

SA represents a popular choice among single-based methods, due to its simplicity, robustness, and its distinctive ability to escape local optima. Over the years, researchers have enhanced SA and hybridized it with other algorithms to boost its efficiency and adaptability. For instance, Zhong et al. [11] incorporated the Metropolis acceptance criterion into a discrete Pigeon-Inspired Optimization algorithm, enabling the pigeons to enhance the algorithm's ability to escape from premature convergence, resulting in strong performance on large-scale TSP instances. Similarly, Ezugwu et al. [12] hybridized SA with Symbiotic Organisms Search, and Zhou et al. [13] used SA to diversify the population of a Gene Expression Programming approach, both studies demonstrating how SA can significantly improve the search dynamics of other metaheuristics. Wu et al. [16] embedded Ant Colony Optimization (ACO) as a search strategy into SA, aiming to solve its problem of slow convergence speed and easily getting stuck in local optimal solutions, achieving faster convergence and superior performance compared with the original ACO and SA algorithms.

Complementing these hybridizations, Zhan et al. [14] introduced a List-Based SA (LBSA) with an adaptive cooling schedule, simplifying parameter tuning and proving highly competitive across standard TSP benchmarks. This ability of being insensitive to the parameter values made the introduced mechanism very attractive and was used in other works. Another interesting work, by Adil and Lakhbab [15], explored the advantage of population-based methods of sharing information of the search space among a population of swarm agents, and proposed a New Improved SA (NISA), which integrates a population-based improvement phase after the Metropolis acceptance step, leveraging social behaviors among candidate solutions to accelerate convergence and improve tour quality.

Parallel to SA-focused enhancements, researchers have explored other advanced metaheuristic frameworks. Recent interesting works include Su et al. [17], where they proposed IDINFO, a discrete variant of the INFO metaheuristic integrated with multi-strategy search and threshold-based 2-opt/3-opt refinements, outperforming GWO, PSO, and AGA on TSPLIB and real-world routing tasks. Another work by Tian et al. [18] introduced a bioinspired two-stage surrogate-assisted algorithm combining clustering and Pigeon-Inspired Optimization that offers effective solutions for high-dimensional TSP instances.

In recent years, Reinforcement Learning (RL) has emerged as a promising paradigm for improving metaheuristics by incorporating adaptive decision-making based on accumulated search experience. Sutton and Barto [23] laid the foundation for Q-learning and other value-based RL methods, which have since been applied to metaheuristic operator selection and parameter tuning. RL-based operator selection enables metaheuristics to learn which neighborhood moves or search strategies are most effective in different phases of the search [24]. Some notable advances include attention-based learning integrated with 2-opt local search [19], learning-guided local search for asymmetric TSP [20]. Reinforcement learning has also been increasingly incorporated into metaheuristics, such as in Q-learning-guided search hybrids [21] and dynamic Q-learning approaches for adaptive annealing [22], enhancing decision-making in complex search spaces.

Collectively, these works illustrate a clear trend: state-of-the-art TSP solvers are increasingly hybrid, adaptive, and learning-guided; an evolution that forms the foundation for our proposed Q-learning-assisted SA.

3. Traveling salesman problem

The Traveling Salesman Problem (TSP) is one of the most prominent and widely studied problems in combinatorial optimization. First formalized by Dantzig and colleagues in 1959 [25], it has attracted significant attention over the years and remains a challenging and highly relevant topic in operations research and computer science. Classified as NP-complete [28], the problem seeks the shortest possible route that visits each of N cities exactly once and returns to the starting point (Hamiltonian cycle). Formally, let $G = (V, E)$ represent a complete graph, where V is the set of cities and $E = \{(c_i, c_j) \mid c_i, c_j \in V, i \neq j\}$ denotes the set of edges connecting them. Each edge (i, j) has an associated travel cost d_{ij}.

An illustrative example of the Traveling Salesman Problem is provided in Figure 1. The left panel shows a Hamiltonian cycle over six cities $c_1, c_2, \ldots, c_6$, where each city is visited exactly once before returning to the starting point.

The right panel represents the same instance as a complete weighted graph, with edge labels indicating the travel costs d_{ij}, the selected tour highlighted in solid black, and non-selected edges shown as dashed lines. The tour

sequence and its total cost are explicitly indicated for clarity. This schematic captures the essential structure of the TSP as a combinatorial optimization problem on a complete weighted graph.

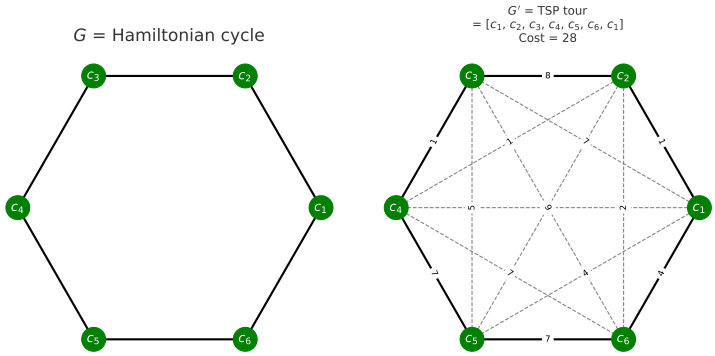


Figure 1. Example of TSP Tour

The TSP can be formulated mathematically as follows:

$$\text{minimize} \quad f(x) = \sum_{i=1}^{N} \sum_{\substack{j=1 \\ j \neq i}}^{N} d_{ij}\, x_{ij} \tag{1}$$

$$\text{subject to} \quad \sum_{\substack{j=1 \\ j \neq i}}^{N} x_{ij} = 1, \qquad \forall i \in \{1, \ldots, N\}, \tag{2}$$

$$\sum_{\substack{i=1 \\ i \neq j}}^{N} x_{ij} = 1, \qquad \forall j \in \{1, \ldots, N\}, \tag{3}$$

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1, \qquad \forall\, S \subset V,\ S \neq \varnothing, \tag{4}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall\, i, j \in \{1, \ldots, N\}. \tag{5}$$

Here, $\mathbf{X}$ represents the search space, x_{ij} is a binary decision variable that equals 1 if edge (i, j) is part of the tour and 0 otherwise. The objective function (1) minimizes the total cost of the route. Constraints (2) and (3) guarantee that each city is visited exactly once, while constraint (4) eliminates the possibility of subtours, ensuring a single continuous tour.

4. Q-learning-Assisted Simulated Annealing

This section describes the proposed hybrid algorithm that integrates Q-learning into the SA framework. After providing a brief introduction to the fundamentals of SA metaheuristic, we proceed to describe the Q-learning mechanism. Lastly, we explain the proposed hybrid method QLSA and its extended version, which allows Q-learning to dynamically guide the search process in SA.

4.1. Simulated Annealing Metaheuristic

SA, first introduced by Kirkpatrick et al. [2], is a widely used probabilistic metaheuristic for solving complex combinatorial optimization problems. The method takes inspiration from the annealing process in metallurgy,

where a material is heated and then cooled slowly to minimize imperfections and reach a stable crystalline state. In an optimization context, this analogy translates into iteratively exploring the solution space while occasionally accepting worse solutions, which helps the algorithm avoid becoming trapped in poor-quality local optima.

Formally, let $\mathcal{S}$ denote the search space and $f : \mathcal{S} \to \mathbb{R}$ the objective function to be minimized.

Starting from an initial solution $s \in \mathcal{S}$, the algorithm generates a candidate solution $s' \in \mathcal{N}(s)$ from its neighborhood. The change in cost is then computed as $\Delta f = f(s') - f(s)$, and the candidate solution is accepted according to the Metropolis criterion defined as follows:

$$P(\Delta f, T) = \begin{cases} 1 & \text{if } \Delta f \leq 0, \\ \exp\big(-\frac{\Delta f}{T}\big) & \text{if } \Delta f > 0 \end{cases} \tag{6}$$

Where T is the current temperature. The temperature is gradually decreased according to a cooling schedule $T \leftarrow \phi(T)$. This mechanism balances exploration (at higher temperatures) and exploitation (as T decreases), enabling SA to gradually refine solutions while preserving search diversity.

In the context of the Traveling Salesman Problem (TSP), the search space $\mathcal{S}$ consists of all possible tours visiting each city exactly once, and the objective function $f(s)$ is the total length of the tour. Neighborhoods are typically generated using *2-opt* or *3-opt* operators, which swap edges or subsequences of the tour. While SA is valued for its ease of use and adaptability, its performance is highly dependent on parameters like the starting temperature, the cooling schedule, and the use of fixed neighborhood operators, which may restrict its flexibility.

In our work, we employ the *2-opt Metropolis operator*, a variant of the classical 2-opt heuristic in which the Metropolis acceptance criterion is incorporated. This modification allows swap moves to be accepted even when they do not yield an immediate improvement, thereby enhancing exploration. The acceptance probability of a swap move is given by the Metropolis criterion 7.

$$P = \exp\left(-\frac{\Delta}{T'}\right) \tag{7}$$

Where T' is the current temperature and Δ represents the change in cost induced by the 2-opt swap. For a swap between edges (x_i, x_{i+1}) and (x_j, x_{j+1}), the cost difference is computed as:

$$\Delta = \big[d(x_i, x_j) + d(x_{i+1}, x_{j+1})\big] - \big[d(x_i, x_{i+1}) + d(x_j, x_{j+1})\big] \tag{8}$$

where $d(a, b)$ denotes the weight (or distance) between nodes a and b.

To enhance its effect, the 2-opt Metropolis operator is applied V times at each iteration, where V is determined as the Hamming distance between the current solution x and the global best solution x^*. This adaptive application strategy increases the intensity of local search when the current solution significantly differs from the global best, while reducing it when the solutions become similar.

The pseudocode for the 2-opt Metropolis operator is given in Algorithm 1.

Algorithme 1: 2-opt Metropolis Operator

Input: Current solution x, global best solution x^*, temperature T', Number of cities n
Output: New solution x'
Compute Hamming distance: $V \leftarrow d_H(x, x^*)$;
for $v \leftarrow 1$ **to** V **do**
  Randomly select an index p;
  for $i \leftarrow p$ **to** $n-1$ **do**
    for $j \leftarrow i+2$ **to** $n-1$ **do**
      Compute cost change:

$$\Delta \leftarrow \big[d(x_i, x_j) + d(x_{i+1}, x_{j+1})\big] - \big[d(x_i, x_{i+1}) + d(x_j, x_{j+1})\big]$$

      if $\Delta < 0$ **then**
        Accept swap and update x by reversing segment $(i+1, j)$;
      end
      else
        Accept swap with probability $P = \exp(-\Delta/T')$;
        if Accepted **then**
          Update x by reversing segment $(i+1, j)$;
        end
      end
    end
  end
end
return x' (updated solution);

When integrated into the SA framework, this operator naturally produces both improving ($\Delta < 0$) and non-improving ($\Delta \geq 0$) solutions, aligning well with the algorithm's search strategy. Moreover, as the acceptance decision depends on the temperature parameter that decreases over iterations, the operator progressively shifts the search toward exploitation in the later stages.

The pseudocode for the SA method is given in Algorithm 2.

Algorithme 2: Classical SA with 2-opt Metropolis

Input: Initial solution x, initial temperature T_0, cooling rate $\alpha \in (0, 1)$, stopping temperature $T_{\min}$
Output: Best-found solution x^*
Initialize best solution: $x^* \leftarrow x$;
Set current temperature: $T \leftarrow T_0$;
while $T > T_{\min}$ **do**
  Generate a neighbor x' by applying a **2-opt Metropolis** move to x;
  Compute cost difference: $\Delta \leftarrow f(x') - f(x)$;
  if $\Delta < 0$ **then**
    Accept neighbor: $x \leftarrow x'$;
  end
  else
    Accept neighbor with probability $P = \exp(-\Delta/T)$;
    if Accepted **then**
      $x \leftarrow x'$;
    end
  end
  if $f(x) < f(x^*)$ **then**
    Update best solution: $x^* \leftarrow x$;
  end
  Update temperature: $T \leftarrow \alpha \cdot T$;
end
return x^*;

4.2. Q-learning Mechanism

Q-learning (QL) [8, 29] is a model-free reinforcement learning algorithm that learns an optimal policy for selecting actions in a Markov Decision Process (MDP) by iteratively refining an action-value function, or Q-function. Unlike model-based methods, it does not require prior knowledge of state transition probabilities, making it particularly suitable for guiding metaheuristics, where problem dynamics are often complex and unpredictable.

The Q-value for a state-action pair (s, a) is updated according to the Bellman equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\big] \tag{9}$$

where $\alpha \in (0,1]$ is the learning rate, $\gamma \in [0,1]$ is the discount factor, r is the immediate reward after taking action a in state s, and s' is the resulting next state. In this framework, the components of the Q-learning algorithms are:

- **States (S):** Represent features of the search process, such as current solution quality, temperature level, or iteration stage.
- **Actions (A):** Define decisions available to the agent, such as selecting a specific neighborhood operator (*2-opt*, *3-opt*) or adjusting algorithm parameters like the cooling rate.
- **Reward (R):** Provides feedback based on the change in the objective value.
- **Policy (π):** Determines how actions are selected.

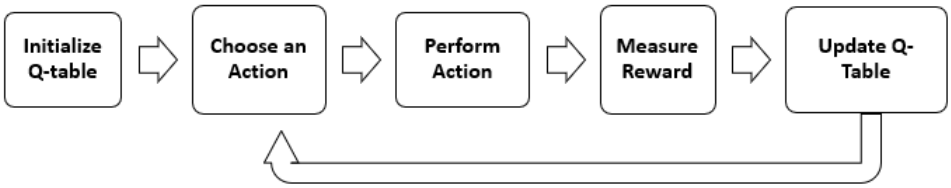


Figure 2. Q-learning process

Figure 2 represents the Q-learning process. This iterative learning process allows the algorithm to identify which actions are most advantageous in different search contexts, adapting its strategy as optimization progresses.

4.3. The Hybrid Approach: Stateless QLSA

The proposed QLSA algorithm integrates a stateless Q-learning mechanism into the classical SA framework, thereby enhancing the adaptivity of the search process. Unlike conventional SA, which explores the neighborhood of a single, fixed solution, QLSA employs Q-learning to dynamically select the most promising candidate solution (action) from a predefined set, based solely on accumulated rewards from previous selections. Once a leader solution is selected, a neighborhood operator is applied. In our implementation, the 2-opt Metropolis move is applied.

This design choice represents a deliberate trade-off between model expressiveness and algorithmic simplicity. The stateless formulation significantly reduces learning complexity and avoids the need for state discretization. Moreover, by learning action utilities aggregated over the entire search trajectory, the proposed approach provides a robust and lightweight adaptive control mechanism that integrates seamlessly into the Simulated Annealing framework without introducing additional hyperparameters or structural dependencies.

Nevertheless, this simplification inherently limits the algorithm's ability to learn context-dependent policies. In particular, the relative usefulness of candidate solutions may vary across different phases of the annealing process (e.g., high-temperature exploration versus low-temperature exploitation), which cannot be explicitly captured by a stateless policy. As such, the proposed QLSA should be viewed as a first-order adaptive mechanism.

In the next section, we will propose a basic stateful version that extends this version.

Let $C = \{c_1, c_2, c_3, c_4\}$ denote the set of candidate actions available for selection at each iteration:

- c_1: the current solution,
- c_2: the global best solution,
- c_3: a randomly generated solution at each iteration that can be seen as a diversity agent,
- c_4: a new solution generated using the double bridge kick operator.

The double-bridge kick consists of selecting four non-adjacent edges in the current tour and reconnecting the resulting segments in a different order, thereby producing a new tour that lies outside the basin of attraction of the current local optimum. Unlike small neighborhood moves, this operator performs a substantial structural modification of the solution while preserving tour feasibility. It was originally introduced in the context of large-step Markov chains for the TSP [31] and later became a standard diversification mechanism with its ability to induce long-range transitions in the solution space while maintaining computational efficiency.

In this stateless formulation, each action $a \in C$ maintains an associated Q-value $Q(a)$, which is iteratively updated according to the following formula:

$$Q(a_t) \leftarrow Q(a_t) + \alpha \left[r_t - Q(a_t) \right] \tag{10}$$

Where α is the learning rate and r_t denotes the immediate reward, defined as the improvement in objective value (i.e., cost reduction) obtained after applying the neighborhood operator to the selected action a_t.
We experiment with two classical policies to resolve the exploration–exploitation dilemma:

- **ε-Greedy**: With probability $1 - \varepsilon$, the policy selects $\arg\max_{a \in C} Q(a)$; with probability ε, it chooses a random action uniformly. This approach is simple, computationally efficient, and widely adopted in Q-learning [23].
- **Softmax (Boltzmann) policy**: Assign a probability to each action:

$$P(a) = \frac{\exp\left(Q(a)/\tau\right)}{\sum_{a' \in C} \exp\left(Q(a')/\tau\right)} \tag{11}$$

  where τ (temperature) governs the exploration intensity. As $\tau \to 0$, softmax behaves greedily; as $\tau \to \infty$, it approximates uniform random selection. This policy is grounded in the Boltzmann distribution tradition and can produce smoother, value-aware exploration than ϵ-greedy. In our work, we use the same temperature as in the SA, $\tau = T$.

These two action-selection policies are widely adopted for balancing exploration and exploitation [30, 23].

The ε-Greedy approach is simple, computationally inexpensive, and provides direct control over the exploration rate through a single parameter, ε, making it particularly effective when the number of available actions is small. In contrast, the Softmax policy offers a value-based probabilistic selection mechanism that naturally smooths the transition between exploration and exploitation. By tuning its temperature parameter analogous to the cooling schedule in SA, Softmax enables a gradual shift towards exploitation without abrupt behavioral changes.

In this study, both policies are employed to examine how different exploration dynamics influence QLSA. Using both allows for a comparison between a more decisive, parameter-driven strategy (ε-Greedy) and a smoother, temperature-controlled one (Softmax), thereby offering deeper insights into the trade-offs between aggressive and tempered exploration in combinatorial optimization. It is worth mentioning that our primary aim in this work is to demonstrate the viability of incorporating Q-learning into SA through simple, well-established policies. Although more advanced exploration methods could potentially enhance performance, their inclusion could also increase algorithmic complexity. The pseudocode for QLSA is provided in Algorithm 3. Also, the corresponding flowchart is shown in Figure 3 for more clarity.

Algorithme 3: Stateless Q-learning-Assisted SA (QLSA)

Input: Initial solution x_0, candidate set size $|C| = 4$, initial temperature T_0, cooling schedule, learning rate α, policy parameters (ε or τ)
Output: Best solution found x^*
Initialize $Q(a) \leftarrow 0$ for all $a \in C$;
Initialize $x \leftarrow x_0$, $x^* \leftarrow x_0$, $T \leftarrow T_0$;
while stopping criterion not met **do**
  Select action $a_t \in C$ using the selected policy (ε-greedy or softmax);
  Apply neighborhood operator (2-opt) to a_t to generate x' with cost $f(x')$;
  Compute reward $r_t \leftarrow f(x) - f(x')$;
  Update Q-value: $Q(a_t) \leftarrow Q(a_t) + \alpha\big[r_t - Q(a_t)\big]$;
  if Metropolis acceptance$(f(x'), f(x), T)$ **then**
    $x \leftarrow x'$;
    if $f(x') < f(x^*)$ **then**
      $x^* \leftarrow x'$;
    end
  end
  Update temperature $T \leftarrow \text{cooling}(T)$;
end
return x^*

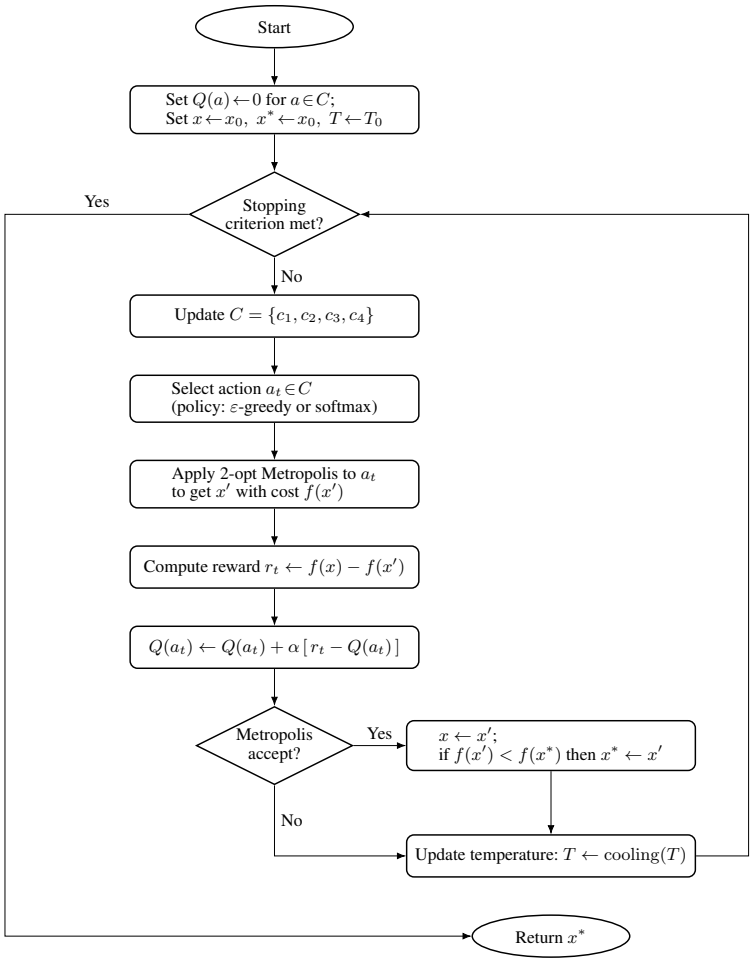


Figure 3. Flowchart of the QLSA algorithm.

4.4. The Hybrid Approach: State-based QLSA (SB-QLSA)

The stateless version of the Q-learning-assisted Simulated Annealing algorithm (QLSA) learns a global preference over candidate leaders without explicitly considering the current phase of the search process. However, in Simulated Annealing, the search dynamics are strongly influenced by the environment's transition from exploration to exploitation. Ignoring this information may limit the learning mechanism's ability to adapt its behavior throughout the optimization process.

To address this limitation, we propose a basic state-based variant of QLSA, denoted SB-QLSA, in which the Q-learning policy is conditioned on the current search state. The objective is to enable the algorithm to learn different leader-selection strategies depending on whether the search is in an exploratory or exploitative phase.

Defining an informative state space for the Q-learning component of QLSA proved to be a nontrivial design choice. A natural option would be to distinguish states based on the temperature level (e.g., high versus low temperature). However, this approach is not well-suited to the present algorithmic setting. In our SA implementation, the temperature follows a strictly decreasing schedule without reheating, meaning that the search deterministically transitions from exploration (high temperature) to exploitation (low temperature). As a consequence, temperature-based states would be largely time-driven and weakly informative, providing little feedback on the actual search dynamics or solution landscape encountered by the algorithm. Thus, we instead define the states based on *solution quality diversity*. Let π_t denote the current solution at iteration t, and let π_t^{best} denote the best solution found so far by the algorithm. The search state is defined based on the Hamming distance between these two solutions. For a problem of size n, the Hamming distance is given by:

$$d_H(\pi_t, \pi_t^{\text{best}}) = \sum_{i=1}^{n} \mathbb{I}\left(\pi_t(i) \neq \pi_t^{\text{best}}(i)\right),$$

where $\mathbb{I}(\cdot)$ is the indicator function. The set of states is defined as

$$\mathcal{S} = \{s_1, s_2\},$$

where s_1 corresponds to a diversified search state and s_2 corresponds to an intensified search state. The current state at iteration t is determined as follows:

$$s(t) = \begin{cases} s_1, & \text{if } d_H(\pi_t, \pi_t^{\text{best}}) > \frac{n}{2}, \\ s_2, & \text{otherwise.} \end{cases}$$

This state definition enables the learning mechanism to distinguish between phases where the current solution is far from the best-known solution and phases where the search is concentrated in its neighborhood.

The action space $\mathcal{A}$ remains identical to that of QLSA and corresponds to the set of candidate leader solutions used to guide neighborhood exploration. In SB-QLSA, the Q-table is defined as a state–action value function:

$$Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}.$$

At each iteration, an action (leader) is selected according to a state-dependent policy (ε-greedy or softmax), and the resulting reward is computed based on the improvement in solution quality, just like in QLSA.

The Q-values are updated using the standard Q-learning update rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_{a' \in \mathcal{A}} Q(s_{t+1}, a') - Q(s_t, a_t) \right],$$

where α is the learning rate, γ is the discount factor, and r_t denotes the reward obtained after selecting action a_t in state s_t. The overall structure of SB-QLSA follows that of QLSA, with the key difference being that both leader selection and Q-value updates are conditioned on the current search state. This allows the learning mechanism to adapt its behavior dynamically as the algorithm transitions from exploration to exploitation, while preserving the simplicity and generality of the original QLSA framework. The pseudo code of SB-QLSA is as follows:

Algorithme 4: State-Based Q-learning-Assisted SA (SB-QLSA)

Input: Initial solution x_0, candidate set size $|C| = 4$, initial temperature T_0, cooling schedule, learning rate α, discount factor γ, policy parameters (ε or τ), diversity threshold δ
Output: Best solution found x^*
State space: $\mathcal{S} = \{0, 1\}$ where 0 = low diversity, 1 = high diversity;
Initialize $Q(s, a) \leftarrow 0$ for all $s \in \mathcal{S}$ and $a \in C$;
Initialize $x \leftarrow x_0$, $x^* \leftarrow x_0$, $T \leftarrow T_0$;
while stopping criterion not met **do**
  Update candidate set $C = \{c_1, c_2, c_3, c_4\}$;
  // Compute current state from solution diversity
  Compute diversity measure D_t using Hamming distance between x and x^*;
  Set $s_t \leftarrow \mathbb{I}[D_t \geq \delta]$;
  Select action $a_t \in C$ using the selected policy on $Q(s_t, \cdot)$ (either ε-greedy or softmax with τ);
  Apply neighborhood operator (2-opt Metropolis) to a_t to generate x' with cost $f(x')$;
  Compute reward $r_t \leftarrow f(x) - f(x')$;
  // Compute next state
  Compute diversity measure D_{t+1} and set $s_{t+1} \leftarrow \mathbb{I}[D_{t+1} \geq \delta]$;
  Update Q-value:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\Big[r_t + \gamma \max_{a \in C} Q(s_{t+1}, a) - Q(s_t, a_t)\Big]$$

  if Metropolis acceptance($f(x'), f(x), T$) **then**
    $x \leftarrow x'$;
    if $f(x') < f(x^*)$ **then**
      $x^* \leftarrow x'$;
    end
  end
  Update temperature $T \leftarrow \text{cooling}(T)$;
end
return x^*;

5. Experimental Results and Discussion

5.1. Benchmark Instances

The proposed QLSA algorithms were evaluated on a set of 17 symmetric Traveling Salesman Problem (TSP) instances from TSPLIB [1] spanning small, medium and moderately large sizes and a mix of Euclidean and geographic metrics.

Table 1 represents each instance and its best-known values for reference.

All experiments were executed on a 64 bit Linux operating system, equipped with an Intel Xeon Gold 6130 CPU at 2.10 GHz, within a virtual machine utilizing the computational resources of HPC-MARWAN (hpc.marwan.ma), provided by the National Center for Scientific and Technical Research (CNRST), Rabat, Morocco. The used programming language is Python 3.11.5 with NumPy/Pandas for data handling. Besides, the TSPLIB95 Python library was used for working with TSPLIB 95 files.

Table 1. TSPLIB instances with size N, distance metric, and Best Known Solution (BKS).

#	Name	N	Metric	BKS
1	ulysses16	16	GEO	6859
2	gr17	17	EXPLICIT	2085
3	ulysses22	22	GEO	7013
4	gr24	24	EXPLICIT	1272
5	bayg29	29	EXPLICIT	1610
6	bays29	29	EXPLICIT	2020
7	dantzig42	42	EXPLICIT	699
8	swiss42	42	EXPLICIT	1273
9	gr48	48	EXPLICIT	5046
10	hk48	48	EXPLICIT	11461
11	eil51	51	EUC_2D	426
12	berlin52	52	EUC_2D	7542
13	st70	70	EUC_2D	675
14	pr76	76	EUC_2D	108159
15	eil76	76	EUC_2D	538
16	rat99	99	EUC_2D	1211
17	eil101	101	EUC_2D	629

5.2. Parameters tuning

To ensure a fair comparison across all variants, the parameters of the simulated annealing (SA) component were held constant. A linear cooling schedule was adopted, defined as follows:

$$T = T_{\max} - [(T_{\max} - T_{\min}) * k]/N_{\max}. \tag{12}$$

where T_k is the temperature at iteration k, T_0 is the initial temperature, $T_{\min}$ is the minimum temperature equal to 0.001, and $N_{\max}$ is the maximum number of iterations. The initial temperature T_0 was calibrated to half the length of the initial tour. Parameter tuning focused exclusively on the Q-learning components. The performance of the variants of our proposed QLSA algorithm is influenced by four tunable parameters. To determine appropriate values for these parameters, the well-known berlin52 instance from the TSPLIB benchmark was selected as the test problem for all tuning experiments. In each experiment, the algorithm was terminated when the maximum number of iterations was reached. The first experiment focused on the learning rate α. This parameter was tested across the range $0.05, 0.1, \ldots, 1$. The corresponding row in Table 2 presents the results for the $QLSA_s$ and $SB-QLSA_s$ variants. The results indicate that the lowest average tour length over 10 runs for $QLSA_s$ was achieved at $\alpha = 0.3$. In contrast, $SB-QLSA_s$ yielded its best average performance at $\alpha = 0.6$. Building on these findings, the second experiment tuned the discount factor γ, tested over the values $0.05, 0.1, \ldots, 1.0$. To isolate the effect of γ, fixed values of α were adopted based on the previous experiment: $\alpha = 0.3$ for $QLSA_s$ and $\alpha = 0.6$ for $SB-QLSA_s$. Analysis of the γ row in Table 2 reveals that the optimal setting differs between the two variants: $\gamma = 1.0$ produced the best performance for $QLSA_s$, whereas $\gamma = 0.8$ was superior for $SB-QLSA_s$.

Table 2. Comparison of mean tour lengths and standard deviations for different values of the learning rate α and discount factor γ in the QLSA$_s$ and SB-QLSA$_s$ variants

Parameters	**Values**	**Mean**		**Std**	
		QLSA$_s$	SB-QLSA$_s$	QLSA$_s$	SB-QLSA$_s$
α	0.05	7803.1	7805.5	130.2	111.0
	0.1	7819.3	7786.4	135.3	168.8
	0.2	7750.8	7792.4	113.1	193.6
	0.3	**7702.1**	7779.7	113.9	118.1
	0.4	7761.5	7850.9	**111.7**	110.4
	0.5	7805.8	7798.8	150.3	145.7
	0.6	7873.7	**7733.7**	136.8	**109.9**
	0.7	7728.4	7739.4	115.5	134.0
	0.8	7834.2	7752.0	135.4	174.8
	0.9	7832.9	7763.4	152.9	153.5
	1	7797.5	7811.7	142.8	121.0
γ	0.05	7796.8	7776.3	110.8	134.4
	0.1	7890.5	7740.6	106.0	134.7
	0.2	7819.7	7749.4	81.2	125.6
	0.3	7864.8	7857.9	145.1	150.6
	0.4	7809.6	7825.9	122.2	173.9
	0.5	7819.2	7762.9	105.1	107.1
	0.6	7795.2	7785.8	135.4	95.4
	0.7	7814.8	7696.9	152.1	83.6
	0.8	7824.0	**7738.9**	136.2	106.7
	0.9	7799.9	7836.0	181.2	151.3
	1	**7778.7**	7819.0	118.3	119.1

The third experiment examined the exploration parameter ϵ (commonly associated with an ϵ-greedy policy), tested in the range 0.05 to 1.0. According to the results reported in the ϵ row of Table 3, both $QLSA_e$ and $SB-QLSA_e$ achieved their lowest average tour lengths when $\epsilon = 1.0$. Finally, the last tuning experiment investigated the Des parameter. Three discrete values were evaluated: 0.001, 0.005, and 0.01. The outcomes summarized in Table 3 demonstrate that Des = 0.001 consistently delivered the best performance across the tested variants.

Table 3. Comparison of mean tour lengths and standard deviations for different values of the learning rate α and discount factor γ in the QLSA$_e$ and SB-QLSA$_e$ variants

Parameters	**Values**	**Mean**		**Std**	
		QLSA$_e$	SB-QLSA$_e$	QLSA$_e$	SB-QLSA$_e$
Epsilon	0.05	8140	8014.1	237.7	303.2
	0.1	8152.2	8169.5	264.27	379.64
	0.2	8004.5	7930.2	201.24	190.31
	0.3	7920.8	7965.6	153.47	199.21
	0.4	7939.3	7951	206.24	248.74
	0.5	7895.5	7937	259.31	189.63
	0.6	7966.9	7861.1	183.38	234.36
	0.7	7873.1	7836.6	194.46	121.76
	0.8	7947.4	7842.7	101.41	203.87
	0.9	7850.9	7902.3	133.70	177.79
	1	**7776.6**	**7796.8**	103.30	189.97
Des	0.001	**7907.7**	**7860.2**	**120.38**	**186.85**
	0.005	8200.5	8050.2	304.01	343.56
	0.01	8266.4	8352.4	654.36	507.00

These tuned parameter values were subsequently adopted for all comparative experiments involving the respective QLSA variants on larger or multiple TSPLIB instances.

5.3. Experimental Setup

In our comparative study, each algorithm was executed multiple times per benchmark instance to ensure a robust statistical evaluation.

Specifically, the main performance table reports results from 10 independent runs for each instance. For each run, we recorded the shortest tour length obtained and the computational time required to achieve this best solution. From these runs, we computed the following statistics for each algorithm and instance pair:

- **Best**: the smallest tour length observed across all runs.
- **Worst**: the largest tour length observed across all runs.
- **Mean**: the average tour length over all runs.
- **Std**: the standard deviation of tour lengths, reflecting result stability.
- **Gap**: the percentage deviation from the Best Known Solution (BKS) reported in TSPLIB.

The **Gap** index is calculated as:

$$\text{Gap}(\%) = \frac{\text{Average} - \text{BKS}}{\text{BKS}} \times 100 \tag{13}$$

where Average is the mean tour length over all runs and BKS is the best-known solution for the corresponding instance. This metric provides a normalized measure of solution quality across instances with different scales. In addition to the **mean gap**, we also report the **minimum gap** (computed from the best tour length found among all runs), allowing a more precise evaluation of each algorithm's peak performance.

Furthermore, to statistically assess the significance of the observed differences in performance, three non-parametric tests were conducted: the **Wilcoxon signed-rank test**, the **Sign test**, and the **Friedman test**. These tests allow for robust comparisons between algorithms without assuming normality in the performance distributions, thus providing more reliable insights into whether observed performance differences are statistically significant.

5.4. Performance Comparison with SA

Following the experimental design and the parameter settings outlined in Sections 5.1 and 5.2, we carried out the experiment on the selected TSP instances from the TSPLIB database.

The detailed results for all tested instances are reported in Table 4.A lower values for the best, worst, and mean metrics reflect a stronger search capability of the algorithm. In the std row, smaller values denote greater stability and consistency in producing high-quality solutions across different runs. Bold values mark the best performance for each metric among the compared algorithms.

Table 4. Comparison statistics between SA, QLSA$_\epsilon$, QLSA$_S$, SB-QLSA$_\epsilon$ and SB-QLSA$_S$

Instances	BK	Stat	SA	QLSA$_S$	QLSA$_\epsilon$	SB-QLSA$_S$	SB-QLSA$_\epsilon$
ulysses16	6859	min	6889	**6859**	**6859**	**6859**	**6859**
		mean	7238.2	**6866.2**	6879.4	6866.9	6880.7
		std	351.398	**5.203**	22.147	12.714	22.146
		max	7886	**6870**	6912	6890	6913
gr17	2085	min	2149	**2085**	**2085**	**2085**	**2085**
		mean	2347.1	2090.5	2092.9	**2090.0**	2090.7
		std	221.352	**6.133**	14.114	10.801	6.783
		max	2746	2103	2130	2120	**2103**
ulyssess22	7013	min	7132	**7013**	**7013**	**7013**	7019
		mean	7670.0	7056.5	7084.1	**7035.1**	7089.4
		std	453.543	39.334	49.088	**38.651**	52.239
		max	8506	7116	7186	**7131**	7170
gr24	1272	min	1286	**1272**	**1272**	**1272**	**1272**
		mean	1481.6	**1290.8**	1297.9	1291.1	1292.0
		std	123.098	18.268	18.941	**16.589**	16.839
		max	1688	1323	1326	1326	**1322**

bayg29	1610	min	1700	**1610**	1620	**1610**	1620
		mean	1897.6	1648.4	1664.9	**1632.7**	1655.6
		std	102.378	26.588	27.188	**23.362**	27.330
		max	2020	1702	1723	**1688**	1690
bays29	2020	min	2095	2028	2026	**2020**	**2020**
		mean	2274.6	2045.0	2060.1	**2041.4**	2059.9
		std	127.509	**18.227**	43.565	21.813	38.083
		max	2465	2082	2164	**2076**	2122
dantzig42	699	min	771	**699**	**699**	713	**699**
		mean	829.7	**714.7**	736.0	730.2	726.9
		std	31.945	11.795	19.905	**9.953**	18.882
		max	882	**740**	771	745	750
swiss42	1273	min	1318	**1286**	**1286**	1287	1288
		mean	1457.6	1323.3	1333.7	**1315.4**	1328.5
		std	86.413	31.263	36.966	**23.899**	27.989
		max	1617	1373	1372	**1366**	1373
gr48	5046	min	5222	5087	5157	**5049**	5058
		mean	5757.4	5215.0	5289.6	**5163.5**	5221.1
		std	325.596	99.962	100.879	**88.295**	116.996
		max	6212	5435	5421	**5345**	5468
hk48	11461	min	**11470**	**11470**	11711	11640	11474
		mean	12944.1	**11809.2**	11974.7	11847.3	11867.3
		std	854.589	179.815	269.270	**142.963**	325.515
		max	14790	12043	12588	**12038**	12654
eil51	426	min	429	432	**427**	430	**427**
		mean	477.1	441.2	442.0	439.6	**438.2**
		std	22.043	**6.033**	9.055	9.902	9.705
		max	501	452	**450**	464	455
berlin52	7542	min	8189	7652	**7590**	7591	7695
		mean	8764.6	**7810.4**	7872.0	7815.9	7894.0
		std	478.212	**126.115**	194.412	140.420	160.868
		max	9636	8087	8152	**7999**	8207
st70	675	min	712	688	**686**	692	698
		mean	752.3	709.5	**707.1**	708.4	726.5
		std	22.85	16.92	16.28	**11.3**	18.8
		max	777	735	731	**726**	759
pr76	108159	min	114938	109733	111952	109990	**108809**
		mean	120732.1	**112449.4**	113407.4	113728.5	113615.6
		std	4032.374	1669.166	**1641.900**	2401.576	3525.411
		max	128401	**114927**	117571	117652	119996
eil76	538	min	565	551	553	**549**	558
		mean	599.9	562.9	566.5	**558.9**	571.8
		std	24.893	9.550	**6.241**	7.838	10.581
		max	640	583	574	**572**	591
rat99	1211	min	1348	1249	**1231**	1241	1273
		mean	1428.9	1298.2	**1282.3**	1286.7	1322.0
		std	71.6	40.9	34.46	**23.8**	56.3
		max	1536	1377	1342	**1315**	1463
eil101	629	min	644	643	**641**	650	652
		mean	686.3	665.9	666.9	**664.4**	670.1
		std	34.42	13.40	18.48	**11.9**	15.37
		max	747	686	701	**682**	700

Table 4 provides a comprehensive comparison between the classical Simulated Annealing (SA) and the various Q-learning-assisted extensions. On the whole, the best average performance (*mean*) is attained by the state-based variants,

specifically SB-QLSA$_S$ and SB-QLSA$_\epsilon$, which outperform the standard SA in all 17 benchmark cases. Notably, SB-QLSA$_S$ achieves the best mean tour length in 8 instances, while the standard QLSA$_S$ remains competitive in smaller problems. This supports the hypothesis that integrating the diversity-based state into the reinforcement learning loop creates a highly efficient search mechanism capable of providing high-quality solutions consistently.

Regarding the best-case performance (*min*), the Q-learning hybrids almost reached or equaled the Best Known (BK) values in almost every instance, whereas the classical SA struggled to converge to the BK in 16 out of 17 cases. Stability-wise, the state-based extensions show a marked improvement; SB-QLSA$_S$ recorded the lowest standard deviation (*std*) in 10 out of 17 instances, proving its superior reliability. This is particularly evident in complex landscapes like *gr48* and *hk48*, where the diversity-aware policy prevents the erratic behavior observed in the classical SA.

Despite SA showing slightly more competitive results on larger instances like *pr76* compared to its own performance on small instances, it still suffers from high variability and significantly higher mean costs. The results indicate that while Q-learning integration is superior in terms of robustness, the SB-QLSA variants mitigate the risk of losing search efficiency on rugged landscapes by adaptively balancing exploration and exploitation through the diversity threshold δ.

Finally, to assess the statistical significance of the observed differences, non-parametric tests were performed. Table 5 reports the results of the statistical comparison between the considered algorithms over all test instances. The Wilcoxon signed-rank test and the Sign test were used to assess pairwise differences in performance, while the Friedman test was applied to compare all methods simultaneously. In the table, "Wins(A)" and "Wins(B)" denote the number of instances on which algorithm A or B achieved better performance, respectively, and the average ranks correspond to the mean Friedman ranks, where lower values indicate better performance.

Table 5. Non parametric statistical comparison

Comparison	**Wilcoxon p**	**Sign p**	**Wins(A)**	**Wins(B)**	**Ties**	**Avg. rank A/B**
QLSA$_s$ vs QLSA$_\epsilon$	**0.0013428**	**0.0023499**	15	2	0	1.824 / 3.235
SA vs QLSA$_\epsilon$	**1.5259e-05**	**1.5259e-05**	0	17	0	5.000 / 3.235
SA vs QLSA$_s$	**1.5259e-05**	**1.5259e-05**	0	17	0	5.000 / 1.824
SA vs SB-QLSA$_s$	**1.5259e-05**	**1.5259e-05**	0	17	0	5.000 / 1.706
SA vs SB-QLSA$_\epsilon$	**1.5259e-05**	**1.5259e-05**	0	17	0	5.000 / 3.235
QLSA$_s$ vs SB-QLSA$_\epsilon$	**7.6294e-05**	**0.00027466**	16	1	0	1.824 / 3.235
QLSA$_\epsilon$ vs SB-QLSA$_s$	**0.0093384**	**0.012726**	3	14	0	3.235 / 1.706
SB-QLSA$_s$ vs SB-QLSA$_\epsilon$	**0.010986**	**0.012726**	14	3	0	1.706 / 3.235
QLSA$_s$ vs SB-QLSA$_s$	0.37782	0.33231	6	11	0	1.824 / 1.706
QLSA$_\epsilon$ vs SB-QLSA$_\epsilon$	1	1	8	9	0	3.235 / 3.235
Friedman $\chi^2 = 48.7529$, $p = 6.5745276e-10$, $N = 17$, $k = 5$						
Avg. ranks (lower is better): SB-QLSA$_s$: 1.706, QLSA$_s$: 1.824, QLSA$_\epsilon$: 3.235, SB-QLSA$_\epsilon$: 3.235, SA: 5.000						

From the table, the Friedman test indicates a statistically significant difference among the algorithms ($\chi^2 = 48.7529$, $p = 6.57 \times 10^{-10}$), confirming that the observed performance variations are unlikely to be due to random fluctuations. The average rank analysis shows that SB-QLSA$_s$ achieves the best overall ranking (1.706), followed closely by QLSA$_s$ (1.824), while QLSA$_\epsilon$ and SB-QLSA$_\epsilon$ obtain intermediate ranks (3.235), and classical SA ranks last (5.000).

Pairwise comparisons further support these observations. Both QLSA variants significantly outperform SA, as indicated by very small p-values in both the Wilcoxon and Sign tests, and by the fact that SA does not outperform any learning-based method on any instance. Similarly, SB-QLSA$_s$ significantly outperforms SA on all instances, confirming the effectiveness of incorporating learning into the search process.

Comparisons among learning-based methods provide additional insight. QLSA$_s$ significantly outperforms QLSA$_\epsilon$ ($p = 0.00134$), suggesting that the softmax-based policy leads to more effective operator selection than the ϵ-greedy strategy in this setting. Likewise, SB-QLSA$_s$ significantly outperforms SB-QLSA$_\epsilon$, indicating that this effect persists in the state-based formulation. More dicussion on policy comparison will be done in the next section.

On the other hand, the comparison between QLSA$_s$ and SB-QLSA$_s$ does not show a statistically significant difference ($p = 0.37782$), suggesting that while the state-based mechanism slightly improves the average rank, its advantage over the stateless variant is not consistently large across all instances. Similarly, no significant difference is observed between QLSA$_\epsilon$ and SB-QLSA$_\epsilon$, indicating comparable behavior for the ϵ-greedy variants.

In summary, the incorporation of state-based Q-learning into simulated annealing yields consistent improvements in both solution quality and run-to-run stability. Among the tested variants, SB-QLSA$_S$ emerges as the most robust overall performer, particularly on more challenging instances, while the ε-greedy strategy (QLSA$_\epsilon$) remains a viable alternative for specific smaller benchmarks like *eil51* and *st70*.

5.5. Policy Comparison: Softmax vs ε-greedy

Tables 6 and 7 report the performance of the QLSA variants using two different action-selection policies: Softmax and ε-greedy.

For each instance, the tables provide the best-known solution (BK), along with the minimum, mean, standard deviation, and maximum tour lengths obtained over multiple independent runs. To allow a normalized comparison across instances of different scales, the mean percentage gap relative to the best-known solution is also reported. Lower gap values indicate better solution quality.

Table 6 presents the results for the stateless QLSA variants, while Table 7 reports the corresponding results for the state-based versions (SB-QLSA). In both tables, the best result for each instance is highlighted in bold. Summary row at the bottom provide the average gap across all instances.

Table 6. Performance comparison of QLSA variants with mean gap to best-known solution (BK).

Instance	BK	stat	QLSA$_S$	Gap(%)	QLSA$_\epsilon$	Gap(%)
ulysses16	6859	min	**6859**	**0.10**	**6859**	0.30
		mean	**6866.2**		6879.4	
		std	**5.20**		22.15	
		max	**6870**		6912	
gr17	2085	min	**2085**	**0.26**	**2085**	0.38
		mean	**2090.5**		2092.9	
		std	**6.13**		14.11	
		max	**2103**		2130	
ulysses22	7013	min	**7013**	**0.62**	**7013**	1.01
		mean	**7056.5**		7084.1	
		std	**39.33**		49.09	
		max	**7116**		7186	
gr24	1272	min	**1272**	**1.48**	**1272**	2.04
		mean	**1290.8**		1297.9	
		std	**18.27**		18.94	
		max	**1323**		1326	
bayg29	1610	min	**1610**	**2.39**	1620	3.41
		mean	**1648.4**		1664.9	
		std	**26.59**		27.19	
		max	**1702**		1723	
bays29	2020	min	2028	**1.24**	**2026**	1.99
		mean	**2045.0**		2060.1	
		std	**18.23**		43.56	
		max	**2082**		2164	
dantzig42	699	min	**699**	**2.25**	**699**	5.29
		mean	**714.7**		736.0	
		std	**11.80**		19.91	
		max	**740**		771	
swiss42	1273	min	**1286**	**3.95**	**1286**	4.77
		mean	**1323.3**		1333.7	
		std	**31.26**		36.97	
		max	1373		**1372**	
gr48	5046	min	**5087**	**3.35**	5157	4.83
		mean	**5215.0**		5289.6	
		std	**99.96**		100.88	
		max	5435		**5421**	
hk48	11461	min	**11470**	**3.04**	11711	4.48
		mean	**11809.2**		11974.7	
		std	**179.81**		269.27	
		max	**12043**		12588	

eil51	426	min mean std max	432 **441.2** **6.03** 452	**3.57**	**427** 442.0 9.06 **450**	3.76
berlin52	7542	min mean std max	7652 **7810.4** **126.11** **8087**	**3.56**	**7590** 7872.0 194.41 8152	4.37
st70	675	min mean std max	688 709.5 16.93 735	5.11	**686** **707.1** **16.29** **731**	**4.76**
pr76	108159	min mean std max	**109733** **112449.4** 1669.17 **114927**	**3.97**	111952 113407.4 **1641.90** 117571	4.85
eil76	538	min mean std max	**551** **562.9** 9.55 583	**4.63**	553 566.5 **6.24** **574**	5.30
rat99	1211	min mean std max	1249 1298.2 40.95 1377	7.20	**1231** **1282.3** **34.46** **1342**	**5.89**
eil101	629	min mean std max	643 **665.9** **13.40** **686**	**5.87**	**641** 666.9 18.48 701	6.03
Average Gap (%)				**3.09**		3.73

Table 7. Performance comparison of SB-QLSA variants with mean gap to BK.

Instance	BK	stat	SB-QLSA$_S$	Gap(%)	SB-QLSA$_\epsilon$	Gap(%)
ulysses16	6859	min mean std max	**6859** **6866.9** **12.71** **6890**	**0.12**	**6859** 6880.7 22.15 6913	0.32
gr17	2085	min mean std max	**2085** **2090.0** 10.80 2120	**0.24**	**2085** 2090.7 **6.78** **2103**	0.27
ulysses22	7013	min mean std max	**7013** **7035.1** **38.65** **7131**	**0.32**	7019 7089.4 52.24 7170	1.09
gr24	1272	min mean std max	**1272** **1291.1** **16.59** 1326	**1.50**	**1272** 1292.0 16.84 **1322**	1.57
bayg29	1610	min mean std	**1610** **1632.7** **23.36**	**1.41**	1620 1655.6 27.33	2.83

		max	**1688**		1690	
bays29	2020	min	**2020**	**1.06**	**2020**	1.97
		mean	**2041.4**		2059.9	
		std	**21.81**		38.08	
		max	**2076**		2122	
dantzig42	699	min	713	4.46	**699**	**4.00**
		mean	730.2		**726.9**	
		std	**9.95**		18.88	
		max	**745**		750	
swiss42	1273	min	**1287**	**3.33**	1288	4.36
		mean	**1315.4**		1328.5	
		std	**23.90**		27.99	
		max	**1366**		1373	
gr48	5046	min	**5049**	**2.33**	5058	3.47
		mean	**5163.5**		5221.1	
		std	**88.30**		117.00	
		max	**5345**		5468	
hk48	11461	min	11640	**3.37**	**11474**	3.54
		mean	**11847.3**		11867.3	
		std	**142.96**		325.51	
		max	**12038**		12654	
eil51	426	min	430	3.19	**427**	**2.86**
		mean	439.6		**438.2**	
		std	9.90		**9.70**	
		max	464		**455**	
berlin52	7542	min	**7591**	**3.63**	7695	4.66
		mean	**7815.9**		7894.0	
		std	**140.42**		160.87	
		max	**7999**		8207	
st70	675	min	**692**	**4.95**	698	7.63
		mean	**708.4**		726.5	
		std	**11.31**		18.88	
		max	**726**		759	
pr76	108159	min	109990	5.15	**108809**	**5.04**
		mean	113728.5		**113615.6**	
		std	**2401.58**		3525.41	
		max	**117652**		119996	
eil76	538	min	**549**	**3.89**	558	6.28
		mean	**558.9**		571.8	
		std	**7.84**		10.58	
		max	**572**		591	
rat99	1211	min	**1241**	**6.25**	1273	9.17
		mean	**1286.7**		1322.0	
		std	**23.90**		56.39	
		max	**1315**		1463	
eil101	629	min	**650**	**5.63**	652	6.53
		mean	**664.4**		670.1	
		std	**11.98**		15.37	
		max	**682**		700	
Average Gap (%)				**2.99**		3.86

The experimental results clearly indicate that the choice of exploration policy has a significant impact on the performance of Q-learning-assisted Simulated Annealing. Across the 17 TSP benchmark instances, the Softmax-based variants consistently outperform their ε-greedy counterparts in both the stateless and state-based formulations.

For the stateless algorithms, QLSA$_S$ achieves a lower average gap to the best-known solutions (3.09%) compared to QLSA$_\epsilon$ (3.73%), and obtains the best mean performance on 15 out of 17 instances. A similar trend is observed for the state-based variants, where SB-QLSA$_S$ achieves an average gap of 2.99%, compared to 3.86% for SB-QLSA$_\epsilon$, with 13 wins out of 17 instances. These improvements are consistent across small, medium, and larger instances, suggesting that the advantage of Softmax is robust with respect to problem size.

The statistical analysis further confirms these observations. From table 5, we can see that both Wilcoxon and Sign tests yield statistically significant differences in both comparisons ($p \prec 0.05$), indicating that the observed performance differences are unlikely to be due to random variation, with QLSA$_S$ winning 15 out of 17 instances and SB-QLSA$_S$ winning 14 in comparison with their greedy-epsilon counterpart.

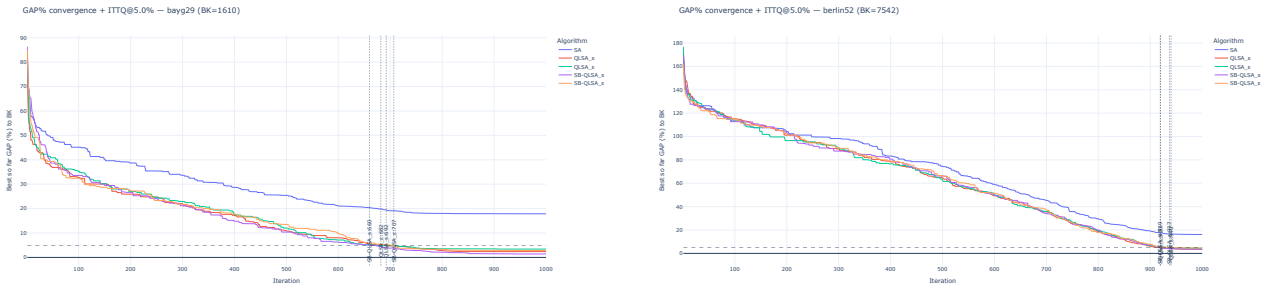
From an algorithmic perspective, these results can be explained by the different exploration mechanisms of the two policies. The ε-greedy policy selects the best action with high probability while exploring uniformly at random with probability ε. Although simple and effective, this strategy does not exploit information about the relative quality of non-best actions, which may lead to less efficient exploration and higher variance in performance. In contrast, the Softmax policy performs a probability-weighted selection based on Q-values, allowing actions with moderately good Q-values to be sampled more frequently than clearly inferior ones. This graded exploration mechanism appears particularly beneficial in the context of QLSA, where candidate solutions often exhibit small but meaningful differences in quality. As a result, Softmax provides a better balance between exploration and exploitation, leading to improved mean performance and lower standard deviations in many instances.

Another noteworthy observation is that the performance gap between policies remains similar in both the stateless and state-based frameworks. This indicates that the benefit of Softmax is largely independent of the introduction of state information and primarily arises from the action selection mechanism itself.

Overall, the results demonstrate that Softmax constitutes a more effective exploration strategy for QLSA than ε-greedy in the tested settings. Its ability to exploit the structure of the learned Q-values leads to more stable convergence and improved solution quality, while maintaining sufficient exploration to avoid premature stagnation.

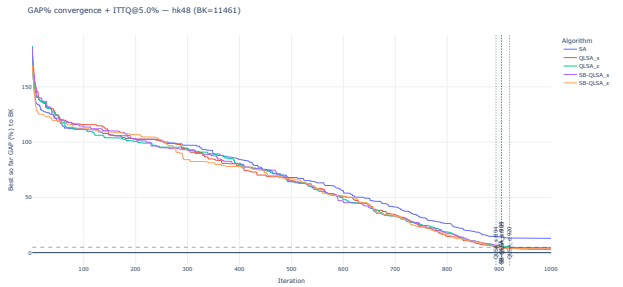
5.6. Convergence of QLSA algorithms

Figures 4 and 5 represent the convergence aspect of the five algorithms for some selected instances. The mean of the gap computed at each iteration over the 10 runs was calculated to efficiently assess the convergence of SA, QLSA$_\epsilon$, QLSA$_s$, SB-QLSA$_\epsilon$ and SB-QLSA$_s$. In addition, the iteration at which a quality target of less than 5% gap is reached is indicated in the plots whenever the target is achieved. The horizontal dashed line represents the target value, while the vertical lines indicate the mean number of iterations required to reach this target.

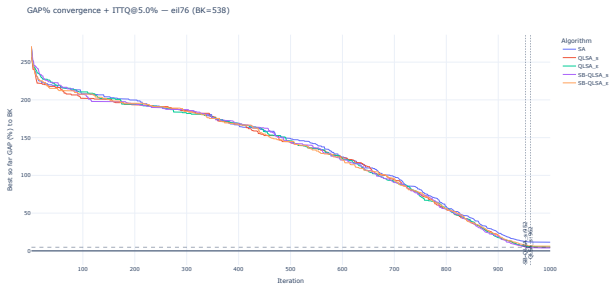


(a) bayg29 (b) berlin52

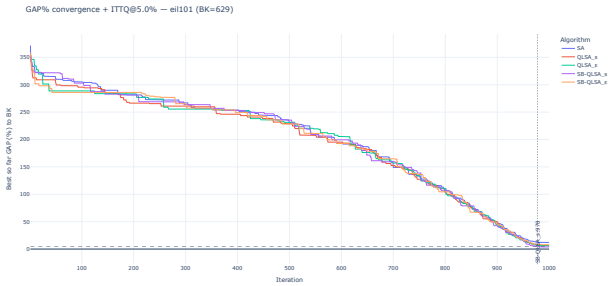
Figure 4. Convergence of the GAP - First Group



(a) hk48



(b) eil76



(c) eil101

Figure 5. Convergence of the GAP - Second Group

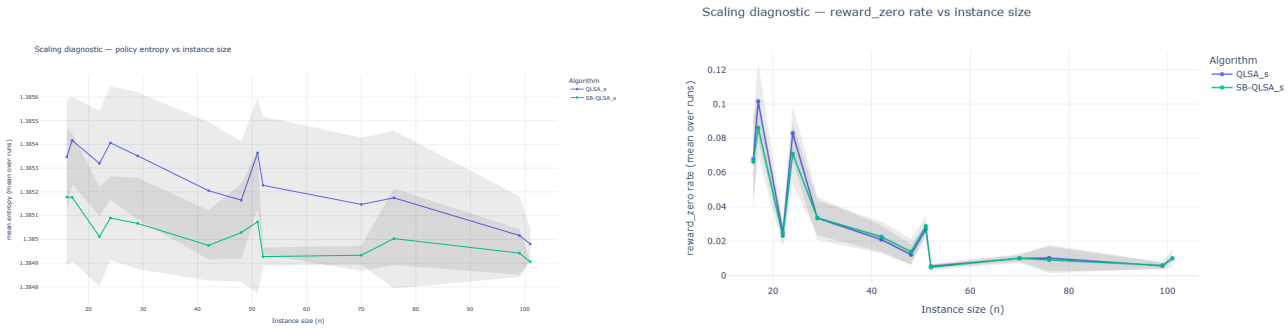
Generally, all algorithms exhibit a rapid reduction of the optimality gap during the early iterations, followed by a slower improvement phase, which is typical of metaheuristic search processes.

On the bayg29 instance, the learning-assisted variants reach the 5% gap target earlier than classical SA, with SB-QLSAs achieving the target in approximately 660 iterations, followed by QLSAϵ and QLSA$_s$, indicating that the learning mechanism can accelerate convergence in small instances.

For berlin52, the differences between algorithms become less pronounced, as all QLSA variants reach the target at similar iteration counts (around 920–940 iterations), suggesting that the benefit of learning remains present but less significant when the landscape becomes more complex. Similar behavior can be seen on hk48, the curves show similar convergence trends among the QLSA variants, with only moderate differences in the iteration required to reach the target, confirming that the advantage of reinforcement-guided selection is instance-dependent and may diminish as problem structure or neighborhood difficulty increases. As for larger instances, the eil76 and eil101 graphs shows that convergence is slower and the target is reached only by SB-QLSA$_s$, indicating that this variant tends to provide more stable improvements in larger search spaces.

Overall, this convergence analysis shows that the proposed learning-assisted variants maintain stable search dynamics and are able to progressively reduce the optimality gap on individual instances. However, convergence behavior alone does not fully characterize the practical performance of the methods. In many real-world applications, problem size increases substantially, and it becomes essential to assess how both solution quality and learning dynamics evolve as the instance dimension grows. In the next paragraph, we analyze whether the reinforcement learning component maintains its ability to discriminate between candidate operators when the search space becomes larger.

Figure 6 reports two aggregate diagnostics computed across instances of increasing size. In order to provide a clear and representative view, we restrict this analysis to the softmax-based variants, which consistently provided the best overall performance in terms of solution quality and stability in the previous experiments.



(a) Scaling entropy vs n

(b) Scaling reward-zero rate vs n

Figure 6. Scalability Analysis graphs

Figure 6a reports the mean *policy entropy* across runs. The policy entropy is defined as

$$H = -\sum_i p(a_i) \log p(a_i), \tag{14}$$

where $p(a_i)$ denotes the probability of selecting action a_i. This quantity measures the dispersion of the action-selection distribution: low entropy indicates strong preference for a subset of candidates, while high entropy corresponds to a more evenly distributed selection policy. The graph indicates that the mean policy entropy remains close to its theoretical maximum for a candidate set of size four. Indeed, the maximum entropy is given by

$$H_{\max} = \log(4) \approx 1.386, \tag{15}$$

and the observed values range approximately between 1.3854 for smaller instances and about 1.3850–1.3852 for larger ones. These results show that the action-selection distribution becomes only weakly differentiated as the instance size increases. In other words, although learning remains stable, the policy exhibits limited differanciation in larger search spaces, reflecting a reduced separation between Q-values.

Figure 6b provides further insight into this behavior by examining the reward signal available to the learning process. The *reward-zero rate*, is defined as the proportion of iterations in which the selected move does not produce an improvement of the incumbent solution. This indicator reflects the frequency of informative learning signals available to the reinforcement learning component.

For small instances such as $n = 16$ or $n = 29$, the reward-zero rate remains at moderate levels, indicating that improving moves occur frequently enough to provide informative feedback to the learning mechanism. As the instance size increases, this rate decreases and stabilizes at values close to zero for instances around $n = 101$. This trend reflects the increasing difficulty of obtaining improving moves in larger TSP instances, where most candidate moves produce marginal or no improvements. Taken together, these results suggest that the reduced policy selectivity observed in larger instances is primarily due to the decreasing magnitude and frequency of informative rewards rather than instability of the learning mechanism itself. As the search space grows, improvements become rarer and smaller in relative terms, which slows down Q-value separation and limits the degree to which the policy can favor specific operators.

Nevertheless, it is important to note that this effect does not prevent the algorithms from maintaining stable convergence behavior, as shown previously. Rather, it highlights an intrinsic scalability challenge of reinforcement-learning-assisted metaheuristics: preserving strong operator discrimination in increasingly large and complex search spaces remains an open research direction, and may require adaptive reward scaling, longer learning horizons, or temperature scheduling strategies.

5.7. Computational time Analysis

Table 8 reports the execution time of SA, QLSA, and SB-QLSA.

Table 8. Computational time comparison (seconds) between SA, QLSA and SB-QLSA variants. The fastest time for each instance is shown in bold.

Instance	SA	QLSA$_s$	QLSA$_\epsilon$	SB-QLSA$_s$	SB-QLSA$_\epsilon$
ulysses16	**8.35**	11.79	9.80	12.00	10.33
gr17	**13.70**	23.22	18.77	23.17	20.17
ulysses22	**30.80**	41.23	39.10	41.11	37.55
gr24	**57.49**	68.89	59.80	66.10	60.59
bayg29	**61.63**	72.65	65.63	70.88	66.08
bays29	**51.22**	57.85	53.16	57.42	53.15
dantzig42	**234.42**	240.51	244.44	244.25	239.97
swiss42	**191.94**	203.46	200.25	201.41	200.55
gr48	**410.85**	455.65	439.89	447.25	440.92
hk48	**365.95**	386.79	373.49	376.99	371.55
eil51	**589.41**	610.57	602.53	611.71	591.92
berlin52	**600.56**	638.66	644.12	645.16	620.02
st70	**2460.60**	2540.81	2498.47	2499.99	2487.54
pr76	3009.84	3112.49	3065.35	3066.64	**2381.83**
eil76	**2379.99**	2466.04	2450.12	2474.23	2485.83
rat99	5027.88	5002.22	5003.58	4981.07	**4915.73**
eil101	151064.69	151560.10	152305.01	**150715.01**	151607.59

As we can see, in terms of computational times, all algorithms remain broadly comparable across the tested instances. Classical SA is generally the fastest method, while QLSA and SB-QLSA introduce a moderate additional cost, which is more noticeable on small instances but decreases as the problem size increases. In several medium and large instances, the differences in execution time become marginal, indicating that the learning mechanism does not significantly affect practical scalability. Moreover, the state-based variants exhibit computational times similar to those of their stateless counterparts, showing that the proposed state representation does not introduce a substantial additional burden.

From a theoretical standpoint, the per-iteration complexity of QLSA remains of the same order as that of classical SA. Let $f(n)$ denote the cost of evaluating a candidate solution of size n. Both algorithms require generating a neighbor solution and computing its objective value, which dominates the runtime and contributes $O(f(n))$ per iteration. The reinforcement learning component adds the selection of an action and the update of a Q-value, operations that require only a small and bounded number of arithmetic computations. Although the maintenance of multiple candidate operators may introduce a small constant overhead in practice, this cost remains independent of the problem size.

Consequently, the overall per-iteration complexity remains $O(f(n))$, meaning that the dominant computational bottleneck in all variants is still the evaluation of candidate solutions. For TSP instances, the most expensive operations remain tour modification and distance evaluation, whose cost depends on the chosen neighborhood operator and typically ranges between $O(1)$ and $O(n)$. In contrast, the reinforcement learning update and policy evaluation remain independent of the instance size and therefore represent a diminishing fraction of the runtime as n increases.

To better quantify the impact of the learning mechanism on runtime, Table 9 reports the relative computational overhead of each variant with respect to classical SA. The results show that the overhead is more pronounced for small instances, where constant-time operations such as policy selection and Q-value updates represent a larger fraction of the total runtime. However, as the instance size increases, the relative overhead decreases rapidly and becomes negligible for large-scale problems. This trend confirms that the dominant computational cost in all algorithms remains the evaluation of candidate solutions and neighborhood operations, whose complexity grows with the problem size. In contrast, the reinforcement learning components introduce only a bounded additional cost that does not scale with the instance dimension. Interestingly, in a few cases the learning-based variants even exhibit slightly lower execution times than SA. This behavior can be attributed to stochastic search trajectories that may reach high-quality regions of the search space earlier, thereby reducing the number of costly evaluations in practice.

Overall, these results indicate that the proposed learning mechanisms affect constant factors but do not alter the practical scalability of the algorithm, making them suitable for medium and large TSP instances where solution quality improvements are obtained at a marginal computational cost.

Table 9. Relative computational overhead (%) of Q-learning-based SA variants compared to SA.

Instance	QLSA$_s$	QLSA$_\epsilon$	SB-QLSA$_s$	SB-QLSA$_\epsilon$
ulysses16	41.20	17.37	43.71	23.71
gr17	69.49	37.01	69.12	47.23
ulysses22	33.86	26.95	33.47	21.92
gr24	19.83	4.02	14.98	5.39
bayg29	17.88	6.49	15.01	7.22
bays29	12.95	3.79	12.11	3.75
dantzig42	2.60	4.28	4.20	2.37
swiss42	6.00	4.32	4.93	4.49
gr48	10.90	7.07	8.86	7.31
hk48	5.69	2.05	3.02	1.53
eil51	3.59	2.22	3.78	0.43
berlin52	6.34	7.25	7.41	3.24
st70	3.26	1.54	1.60	1.10
pr76	3.41	1.84	1.88	-20.86
eil76	3.62	2.95	3.95	4.44
rat99	-0.51	-0.48	-0.93	-2.22
eil101	0.33	0.82	-0.23	0.36

These observations suggest that the trade-off between solution quality and computational effort remains favorable, as the additional runtime required by the learning mechanisms becomes negligible compared to the cost of evaluating candidate solutions for large instances.

6. Conclusion and Future Work

This paper investigated multiple variants of reinforcement-learning-assisted variants of simulated annealing for solving the Traveling Salesman Problem. Several Q-learning-based strategies were proposed and analyzed, including stateless and state-based formulations, as well as different action-selection policies. Among these variants, the proposed State-Based Q-Learning-Assisted Simulated Annealing (SB-QLSA), particularly when combined with a softmax selection policy, achieved the most consistent and robust performance across the tested benchmark instances. Extensive experiments on a wide range of TSPLIB instances show that Q-learning-assisted variants significantly improve solution quality and robustness compared with classical simulated annealing. Statistical analyses, including Friedman, Wilcoxon, and Sign tests, confirm that these improvements are statistically significant which indicate that incorporating learning mechanisms to guide the selection of neighborhood leaders enhances both the stability and effectiveness of the search process, while the introduction of state information further improves the algorithm's ability to adapt its behavior to different search regimes.

The experimental analysis also provided insight into the scalability of the proposed methods. While all learning-assisted variants exhibit stable convergence behavior and strong performance on small and medium-sized instances, a gradual decrease in performance is observed as the instance size increases. Diagnostic analyses indicate that informative rewards become less frequent and smaller in magnitude in larger search spaces, which slows down the separation of action values and reduces policy selectivity. This behavior reflects intrinsic challenges of applying reinforcement learning within large combinatorial landscapes rather than instability of the learning process itself.

Several factors contribute to this limitation. The stateless formulation restricts the ability of the learning mechanism to adapt its guidance strategy to different phases of the annealing process, while the use of a fixed and compact candidate set may provide insufficient diversity for effective navigation of large and complex landscapes. In addition, learning parameters that are effective for small and medium-sized instances may require careful scaling or adaptation to fully exploit reinforcement learning on larger problems.

Beyond the TSP, the proposed framework remains inherently problem-independent. By coupling a metaheuristic backbone with a reinforcement learning mechanism to guide neighborhood exploration, the approach can be extended to other combinatorial optimization problems with minimal modification, requiring only the definition of problem-specific neighborhood operators. Potential applications include vehicle routing, scheduling, assignment, Multidimensional Knapsack Problem and other large-scale discrete optimization problems.

Future work will focus on improving scalability and adaptability. In particular, adaptive and self-tuning parameter control strategies will be investigated to achieve a more effective balance between exploration and exploitation in large-scale instances. Extending the learning mechanism toward richer state representations and context-sensitive reinforcement learning models represents a promising direction for enhancing phase-dependent decision-making. Additional improvements may also be obtained through dynamic candidate-set generation, hybridization with complementary metaheuristics, and

parallel implementations. Finally, evaluating the proposed variants on much larger TSP instances and other combinatorial optimization problems constitutes a natural continuation of this work.

REFERENCES

1. G. Reinelt, *TSPLIB—A traveling salesman problem library* , ORSA Journal on Computing, vol. 3, no. 4, pp. 376-384, 1991.
2. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, *Optimization by Simulated Annealing*, Science, vol. 220, no. 4598, pp. 671–680, 1983.
3. E. G. Talbi, *Metaheuristics: from design to implementation*, John Wiley & Sons, 2009.
4. I. H. Osman and G. Laporte, *Metaheuristics: A bibliography*, Annals of Operations Research, vol. 63, no. 5, pp. 513–623, 1996.
5. V. Granville, M. Krivanek, and J.-P. Rasson, *Simulated Annealing: A Proof of Convergence*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 16, no. 6, pp. 652–656, 1994.
6. M. Tessari and G. Iacca, *Reinforcement Learning Based Adaptive Metaheuristics*, in GECCO '22 Companion Proceedings, p. 1854-1861, 2022.
7. R.K. Kincaid and A. Ninh, *Simulated Annealing* , in: Martí, R., Martínez-Gavara, A. (Eds.), Discrete Diversity and Dispersion Maximization: A Tutorial on Metaheuristic Optimization. Springer International Publishing, Cham, pp. 221–249, 2023.
8. C. J. C. H. Watkins and P. Dayan, *Q-learning*, Machine Learning, vol. 8, no. 3, pp. 279–292, 1992.
9. İ. İlhan, *An Improved Simulated Annealing Algorithm with Crossover Operator for Capacitated Vehicle Routing Problem*, Swarm and Evolutionary Computation, vol. 64, art. 100911, 2021.
10. R. Reijnen, Y. Zhang, H. C. Lau, and Z. Bukhsh, *Online Control of Adaptive Large Neighborhood Search Using Deep Reinforcement Learning*, in In : Proceedings of the International Conference on Automated Planning and Scheduling, p. 475-483, 2024.
11. Y. Zhong, L. Wang, M. Lin and H. Zhang, *Discrete pigeon-inspired optimization algorithm with Metropolis acceptance criterion for large-scale traveling salesman problem*, Swarm and Evolutionary Computation, vol. 4, pp. 134–144, 2019.
12. A. E. Ezugwu, A. O. Adewumi, and M. E. Frîncu, *Simulated annealing based symbiotic organisms search optimization algorithm for traveling salesman problem* , Expert Systems with Applications, vol. 77, pp. 189–210, 2017.
13. Zhou, A.-H., Zhu, L.-P., Hu, B., Deng, S., Song, Y., Qiu, H., and Pan, S, *Traveling-Salesman-Problem Algorithm Based on Simulated Annealing and Gene-Expression Programming*, Information, vol. 10, n. 1, 7.
14. Zhan, S. H., Lin, J., Zhang, Z. J., and Zhong, Y. W., *List-Based Simulated Annealing Algorithm for Traveling Salesman Problem*, Computational Intelligence and Neuroscience, vol. 2016, n. 1, pp. 1712630, 2016.
15. A. Adil and H. Lakhbab, *A new improved simulated annealing for traveling salesman problem*, Mathematical modeling and computing, , vol. 10, no 3, p. 764-771, 2023.
16. Y. Wu, H. Wang, M. Li, H. Tan, D. Wang, and M. Sheng, *The adaptive two-stage ant colony simulated annealing algorithm for solving the Traveling Salesman Problem*, RAIRO-Operations Research, vol. 59, no. 2, pp. 1199–1213, 2025.
17. Y. Su, Y. Ran, Z. Yan, Y. Zhang, and X. Yang, *Solving the Traveling Salesman Problem Using the IDINFO Algorithm*, ISPRS International Journal of Geo-Information, vol. 14, n. 3, p. 111, 2025.
18. A. Q. Tian, H. X. Lv, X. Y. Wang,J. S. Pan and V. Snášel, *Bioinspired Discrete Two-Stage Surrogate-Assisted Algorithm for Large-Scale Traveling Salesman Problem*, Journal of Bionic Engineering, p. 1-14, 2025.
19. W. Kool, H. van Hoof, and M. Welling, *Attention, Learn to Solve Routing Problems!*, arXiv preprint arXiv:1803.08475, 2018.
20. L. Zhou, Y. Ju, and S. Moura, *Learning-Guided Local Search for Asymmetric Traveling Salesman Problem*, 2nd AI for Math Workshop @ ICML 2025.
21. S. H. Rahaman, S. Mondal, and M. K. Maiti, *A collaborative use of two metaheuristic methods using Q-learning for the travelling salesman problems in different fields: an application to smart home delivery system*, Evolutionary Intelligence, vol. 18, n. 3, p. 53, 2025.
22. L. Sun, M. Zhu, J. Li and H. Zhang, *Adaptive annealing dynamic Q-learning based on subroute optimization for solving the traveling salesman problem*, Cluster Computing, vol. 28, n. 7, p. 454, 2025.
23. R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, 2018.
24. A. Fialho, L. Da Costa, M. Schoenauer, and M. Sebag, *Analyzing Bandit-Based Adaptive Operator Selection*, Annals of Mathematics and Artificial Intelligence, vol. 60, no. 1, pp. 25–64, 2010.
25. Dantzig, G. B., Fulkerson, D. R., and Johnson, S. M. , *On a Linear-Programming, Combinatorial Approach to the Traveling-Salesman Problem* , Operations Research, vol. 7, no 1, p. 58-66, 1959.
26. Glover, F., *Tabu search—part I* , ORSA Journal on computing, vol. 1, no. 3, pp. 190–206, 1989.
27. Kennedy, J., and Eberhart, R *Particle swarm optimization* , In Proceedings of ICNN'95-international conference on neural networks, vol. 4, no. 3, pp. 1942-1948, 1995.
28. BRUCKER, Peter, *NP-Complete operations research problems and approximation algorithms* , Zeitschrift für Operations-Research, vol. 23, no 3, p. 73-94, 1979
29. B. Jang, M. Kim, G. Harerimana and J. W. Kim, *Q-Learning Algorithms: A Comprehensive Classification and Applications*, in IEEE Access, vol. 7, pp. 133653-133667, 2019
30. Tokic, M., Palm, G., *Value-Difference Based Exploration: Adaptive Control between Epsilon-Greedy and Softmax*, In: Bach, J., Edelkamp, S. (eds) KI 2011: Advances in Artificial Intelligence. KI 2011. Lecture Notes in Computer Science(), vol 7006. Springer, Berlin, Heidelberg, 2011.
31. Martin, O., Otto, S. W., and Felten, E. W., *Large-step Markov chains for the traveling salesman problem*, Oregon Graduate Institute of Science and Technology, Department of Computer Science and Engineering. pp. 299-326, 1991.